Ex. No: 1 WRITE A CODE TO IMPLEMENT AES ENCRYPTION AND DECRYPTION

PROGRAM:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class AESExample {

    // Method to encrypt a plain text using AES algorithm
    public static String encrypt(String plainText, String secretKey) throws Exception {
        // Create a Cipher instance with AES algorithm
        Cipher cipher = Cipher.getInstance("AES");

        // Convert the secret key to byte array
        byte[] key = secretKey.getBytes("UTF-8");

        // Create a SecretKeySpec using the byte array
        SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");

        // Initialize the cipher with encryption mode and the secret key
        cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec);

        // Encrypt the plain text
        byte[] encryptedText = cipher.doFinal(plainText.getBytes("UTF-8"));

        // Return the encrypted text as a Base64 encoded string
        return Base64.getEncoder().encodeToString(encryptedText);
    }

    // Method to decrypt an encrypted text using AES algorithm
    public static String decrypt(String encryptedText, String secretKey) throws Exception {
        // Create a Cipher instance with AES algorithm
        Cipher cipher = Cipher.getInstance("AES");

        // Convert the secret key to byte array
        byte[] key = secretKey.getBytes("UTF-8");
```

```
        // Create a SecretKeySpec using the byte array
        SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");

        // Initialize the cipher with decryption mode and the secret key
        cipher.init(Cipher.DECRYPT_MODE, secretKeySpec);

        // Decode the encrypted text from Base64
        byte[] decodedText = Base64.getDecoder().decode(encryptedText);

        // Decrypt the text
        byte[] decryptedText = cipher.doFinal(decodedText);

        // Return the decrypted text as a string
        return new String(decryptedText, "UTF-8");
    }

    public static void main(String[] args) {
        try {
            // Example key (must be 16, 24, or 32 bytes long)
            String secretKey = "1234567890123456"; // 16 bytes key

            // Plain text to encrypt
            String plainText = "Hello, World!";

            // Encrypt the plain text
            String encryptedText = encrypt(plainText, secretKey);
            System.out.println("Encrypted Text: " + encryptedText);

            // Decrypt the encrypted text
            String decryptedText = decrypt(encryptedText, secretKey);
            System.out.println("Decrypted Text: " + decryptedText);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:

Encrypted Text: s1aiR0qHAayxg11CyTDX1Q==
Decrypted Text: Hello, World!

Ex. No: 2 WRITE A CODE TO IMPLEMENT AES ENCRYPTION AND DECRYPTION

DSA ALGORITHM:

```
import javax.crypto.KeyAgreement;

import java.security.*;

import java.security.spec.X509EncodedKeySpec;

import java.util.Base64;

public class DiffieHellmanExample {

    public static void main(String[] args) throws Exception {

        // Generate key pairs for Alice

        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DH");

        keyPairGen.initialize(2048);

        KeyPair aliceKeyPair = keyPairGen.generateKeyPair();

        // Generate key pairs for Bob

        KeyPair bobKeyPair = keyPairGen.generateKeyPair();

        // Alice generates shared secret

        KeyAgreement aliceKeyAgree = KeyAgreement.getInstance("DH");

        aliceKeyAgree.init(aliceKeyPair.getPrivate());

        aliceKeyAgree.doPhase(bobKeyPair.getPublic(), true);

        byte[] aliceSharedSecret = aliceKeyAgree.generateSecret();

        System.out.println("Alice's Shared Secret: " +
    Base64.getEncoder().encodeToString(aliceSharedSecret));

        // Bob generates shared secret
```

```
        KeyAgreement bobKeyAgree = KeyAgreement.getInstance("DH");
        bobKeyAgree.init(bobKeyPair.getPrivate());
        bobKeyAgree.doPhase(aliceKeyPair.getPublic(), true);
        byte[] bobSharedSecret = bobKeyAgree.generateSecret();
        System.out.println("Bob's Shared Secret: " + Base64.getEncoder().encodeToString(bobSharedSecret));
        // Check if both secrets match
        if (MessageDigest.isEqual(aliceSharedSecret, bobSharedSecret)) {
            System.out.println("Shared secrets are identical!");
        } else {
            System.out.println("Shared secrets do not match.");
        }
    }
}
```

RSA PROGRAM:

```
import java.security.*;
import javax.crypto.Cipher;
import java.util.Base64;

public class RSAExample {

    // Method to generate RSA key pair
    public static KeyPair generateKeyPair() throws NoSuchAlgorithmException {
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(2048);  // RSA key size
        return keyGen.generateKeyPair();
    }

    // Method to encrypt a message using the public key
    public static String encrypt(String message, PublicKey publicKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedBytes = cipher.doFinal(message.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    // Method to decrypt a message using the private key
    public static String decrypt(String encryptedMessage, PrivateKey privateKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedMessage));
        return new String(decryptedBytes);
    }

    public static void main(String[] args) {
        try {
            // Generate RSA Key Pair
            KeyPair keyPair = generateKeyPair();
            PublicKey publicKey = keyPair.getPublic();
            PrivateKey privateKey = keyPair.getPrivate();

            // Message to be encrypted
            String message = "Hello, RSA!";
            String encryptedMessage = encrypt(message, publicKey);
            System.out.println("Encrypted Message: " + encryptedMessage);
            String decryptedMessage = decrypt(encryptedMessage, privateKey);
            System.out.println("Decrypted Message: " + decryptedMessage);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }}
```

DSA OUTPUT:

Alice's Shared Secret: kIUzBzQiX6r0FJ7hg1KmMVSKdrD2S1tAnKm93JwLXtY=

Bob's Shared Secret: kIUzBzQiX6r0FJ7hg1KmMVSKdrD2S1tAnKm93JwLXtY=

Shared secrets are identical!

RSA OUTPUT:

Encrypted Message:
Xy6VpNmdwFDqldz8hNSCsUMJc6cQVnpT+eUc84oWE6AolRd4kH7rdzKDyw2WbvO
+/0kc8mUJz9XYRmHDF+1FQ==
Decrypted Message: Hello, RSA!

Ex. No: 3 IMPLEMENT DIGITAL SIGNATURE USING RSA AND SHA ALGORITHM

PROGRAM:

```
import java.security.*;
import java.util.Base64;

public class DigitalSignatureExample {

    // Method to generate RSA key pair
    public static KeyPair generateKeyPair() throws NoSuchAlgorithmException {
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(2048);  // RSA key size
        return keyGen.generateKeyPair();
    }

    // Method to create a digital signature for the message
    public static String signMessage(String message, PrivateKey privateKey) throws Exception {
        // Get a Signature instance for SHA256withRSA
        Signature signature = Signature.getInstance("SHA256withRSA");

        // Initialize the signature with the private key
        signature.initSign(privateKey);

        // Provide the data to be signed
        signature.update(message.getBytes("UTF-8"));

        // Sign the data and return the signature as Base64 encoded string
        byte[] digitalSignature = signature.sign();
        return Base64.getEncoder().encodeToString(digitalSignature);
    }

    // Method to verify the digital signature
    public static boolean verifySignature(String message, String signatureToVerify, PublicKey publicKey)
     throws Exception {
        // Get a Signature instance for SHA256withRSA
        Signature signature = Signature.getInstance("SHA256withRSA");

        // Initialize the signature with the public key
        signature.initVerify(publicKey);

        // Provide the data whose signature needs to be verified
        signature.update(message.getBytes("UTF-8"));

        // Verify the signature
        byte[] signatureBytes = Base64.getDecoder().decode(signatureToVerify);
```

```
        return signature.verify(signatureBytes);
    }

    public static void main(String[] args) {
        try {
            // Generate RSA Key Pair
            KeyPair keyPair = generateKeyPair();
            PublicKey publicKey = keyPair.getPublic();
            PrivateKey privateKey = keyPair.getPrivate();

            // Message to be signed
            String message = "This is a confidential message.";

            // Create the digital signature for the message
            String digitalSignature = signMessage(message, privateKey);
            System.out.println("Digital Signature: " + digitalSignature);

            // Verify the digital signature
            boolean isVerified = verifySignature(message, digitalSignature, publicKey);
            System.out.println("Signature verification: " + isVerified);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:

Digital Signature:
mRtbgn9z4jZWhPwrXgiM5KPxo5ZRZyYx8xD4bbgPFaMd1IoWIV1w7W8JDb6dUKY0d7ObExNj7I
T7K6BhHjP1ClJj0DZ23TPyHMb/wXGSP6XjOnfN0Udz2BaPAwMkqT82YfGltc1EkFRZqGzAowmn
+FoFOx7M13acmlFmrgVZ+LWE=

Signature verification: true

Ex. No: 4 CREATING MERKLE TREE

PROGRAM:

```
import java.security.MessageDigest;
import java.util.ArrayList;
import java.util.List;
import java.util.Base64;

public class MerkleTree {

   // Method to generate the hash of a given data
   public static String hash(String data) throws Exception {
      MessageDigest digest = MessageDigest.getInstance("SHA-256");
      byte[] hash = digest.digest(data.getBytes("UTF-8"));
      return Base64.getEncoder().encodeToString(hash);
   }

   // Recursive method to build the Merkle root from a list of transactions
   public static String buildMerkleTree(List<String> leaves) throws Exception {
      // If only one leaf remains, this is the root hash
      if (leaves.size() == 1) {
         return leaves.get(0);
      }

      List<String> newLevel = new ArrayList<>();

      // Process pairs of leaves
      for (int i = 0; i < leaves.size(); i += 2) {
         String left = leaves.get(i);
         String right = (i + 1 < leaves.size()) ? leaves.get(i + 1) : leaves.get(i);  // If odd, duplicate the last hash
         String combinedHash = hash(left + right);  // Concatenate and hash the two child nodes
         newLevel.add(combinedHash);
      }

      // Recursively build the Merkle tree with the new level of combined hashes
      return buildMerkleTree(newLevel);
   }

   public static void main(String[] args) throws Exception {
      // Sample list of transactions (leaves of the Merkle tree)
      List<String> transactions = new ArrayList<>();
      transactions.add("tx1");
      transactions.add("tx2");
```

```
        transactions.add("tx3");
        transactions.add("tx4");
        transactions.add("tx5");

        // Hash the transactions (initial leaf nodes)
        List<String> hashedTransactions = new ArrayList<>();
        for (String transaction : transactions) {
            hashedTransactions.add(hash(transaction));
        }

        System.out.println("Hashed Transactions (Leaves of Merkle Tree):");
        for (String hashedTransaction : hashedTransactions) {
            System.out.println(hashedTransaction);
        }

        // Build Merkle root
        String merkleRoot = buildMerkleTree(hashedTransactions);
        System.out.println("\nMerkle Root: " + merkleRoot);
    }
}
```

OUTPUT:

Hashed Transactions (Leaves of Merkle Tree):
cJtVvT2g9ag4ElvQ7iDFv918q6FzkS1CgcroFreaIBs=
J8pkwJKpWcftxSXtRehFsd5qdZDRc/0vrZEzyKd5oeM=
HzyxjoliVtfWu4wRpuxx8AXHXeBeOb6uXZO70eLIt6k=
QbY3z9nrPi9g9zT5ykTlwVWcb0gdSdbtaJHz6aCGrHg=
qMDM6LsGfpHPJ2bCa+Tl18+6PTMj3BnQioNDkaHOWs8=

Merkle Root: YkKDnu4nQL6eS3oEM8lqzarvkOWme6a1VpV4CE+z8QQ=

Ex. No: 5 CREATIONS OF BLOCK

PROGRAM:

```
import java.security.MessageDigest;
import java.util.Date;

class Block {
    private String hash;
    private String previousHash;
    private String data;
    private long timestamp;
    private int nonce;

    // Constructor for the Block
    public Block(String data, String previousHash) {
        this.data = data;
        this.previousHash = previousHash;
        this.timestamp = new Date().getTime();
        this.hash = calculateBlockHash();  // Calculate the hash when the block is created
    }

    // Calculate the hash for the block based on its content
    public String calculateBlockHash() {
        String dataToHash = previousHash + Long.toString(timestamp) + Integer.toString(nonce) + data;
        MessageDigest digest;
        byte[] bytes = null;
        try {
            digest = MessageDigest.getInstance("SHA-256");
            bytes = digest.digest(dataToHash.getBytes("UTF-8"));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        StringBuffer buffer = new StringBuffer();
        for (byte b : bytes) {
            buffer.append(String.format("%02x", b));
        }
        return buffer.toString();
    }

    // Mine the block by adjusting the nonce until a hash is generated that matches the difficulty
    public void mineBlock(int difficulty) {
        String target = new String(new char[difficulty]).replace('\0', '0');  // Create a string with leading
     'difficulty' zeros
```

```
        while (!hash.substring(0, difficulty).equals(target)) {
            nonce++;
            hash = calculateBlockHash();
        }
        System.out.println("Block Mined! Hash: " + hash);
    }

    // Getter methods
    public String getHash() {
        return hash;
    }

    public String getPreviousHash() {
        return previousHash;
    }

    public String getData() {
        return data;
    }
}

public class Blockchain {
    public static void main(String[] args) {
        int difficulty = 4;  // Difficulty level for proof-of-work (number of leading zeros required in hash)

        // Creating the Genesis Block (the first block in the chain)
        Block genesisBlock = new Block("First block data", "0");
        System.out.println("Mining Genesis Block...");
        genesisBlock.mineBlock(difficulty);

        // Creating the second block
        Block secondBlock = new Block("Second block data", genesisBlock.getHash());
        System.out.println("Mining Second Block...");
        secondBlock.mineBlock(difficulty);

        // Creating the third block
        Block thirdBlock = new Block("Third block data", secondBlock.getHash());
        System.out.println("Mining Third Block...");
        thirdBlock.mineBlock(difficulty);
    }
}
```

OUTPUT

```
Mining Genesis Block...
Block Mined! Hash: 0000be869c69467a06a1297501a1319679c4fca58ccc6848ec651b9a44c04670
Mining Second Block...
Block Mined! Hash: 000041a77db0c0143859f8fcc5fa10eb99808d995fedd2845df59b665d26f78d
Mining Third Block...
Block Mined! Hash: 0000a0fa3f2116e6eb437329c01bb48e9ad976748631196dcdbd7f07b3a37beb
```

Ex. No: 6 BLOCK CHAIN IMPLEMENTATION

PROGRAM:

```
import java.security.MessageDigest;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

// Block class representing a single block in the blockchain
class Block {
    public String hash;
    public String previousHash;
    private String data;
    private long timestamp;
    private int nonce;

    // Constructor for the Block
    public Block(String data, String previousHash) {
        this.data = data;
        this.previousHash = previousHash;
        this.timestamp = new Date().getTime();
        this.hash = calculateBlockHash();  // Calculate the block's hash upon creation
    }

    // Method to calculate the block's hash using SHA-256
    public String calculateBlockHash() {
        String dataToHash = previousHash + Long.toString(timestamp) + Integer.toString(nonce) + data;
        MessageDigest digest;
        byte[] bytes = null;
        try {
            digest = MessageDigest.getInstance("SHA-256");
            bytes = digest.digest(dataToHash.getBytes("UTF-8"));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        StringBuffer buffer = new StringBuffer();
        for (byte b : bytes) {
            buffer.append(String.format("%02x", b));
        }
        return buffer.toString();
    }

    // Method to mine the block by finding a valid hash that satisfies the difficulty level
    public void mineBlock(int difficulty) {
```

```
        String target = new String(new char[difficulty]).replace('\0', '0');  // Create a target string with leading zeros
        while (!hash.substring(0, difficulty).equals(target)) {
            nonce++;
            hash = calculateBlockHash();
        }
        System.out.println("Block Mined: " + hash);
    }
}

// Blockchain class to manage the chain of blocks
class Blockchain {
    public static List<Block> blockchain = new ArrayList<>();
    public static int difficulty = 4;  // Difficulty level for mining

    // Method to add a new block to the blockchain
    public static void addBlock(Block newBlock) {
        newBlock.mineBlock(difficulty);
        blockchain.add(newBlock);
    }

    // Method to check the validity of the blockchain
    public static boolean isChainValid() {
        Block currentBlock;
        Block previousBlock;

        // Loop through all blocks and check their hashes
        for (int i = 1; i < blockchain.size(); i++) {
            currentBlock = blockchain.get(i);
            previousBlock = blockchain.get(i - 1);

            // Check if current block's hash is valid
            if (!currentBlock.hash.equals(currentBlock.calculateBlockHash())) {
                System.out.println("Current Block's hash is invalid");
                return false;
            }

            // Check if previous block's hash matches the stored hash in the current block
            if (!currentBlock.previousHash.equals(previousBlock.hash)) {
                System.out.println("Previous Block's hash is invalid");
                return false;
            }
        }

        return true;
    }
}

// Main class to run the blockchain implementation
```

```
public class Main {
    public static void main(String[] args) {
        // Create and add the Genesis Block (the first block)
        Block genesisBlock = new Block("First block data", "0");
        System.out.println("Mining Genesis Block...");
        Blockchain.addBlock(genesisBlock);

        // Add second block
        Block secondBlock = new Block("Second block data",
     Blockchain.blockchain.get(Blockchain.blockchain.size() - 1).hash);
        System.out.println("Mining Second Block...");
        Blockchain.addBlock(secondBlock);

        // Add third block
        Block thirdBlock = new Block("Third block data",
     Blockchain.blockchain.get(Blockchain.blockchain.size() - 1).hash);
        System.out.println("Mining Third Block...");
        Blockchain.addBlock(thirdBlock);

        // Verify the validity of the blockchain
        System.out.println("\nBlockchain is valid: " + Blockchain.isChainValid());

        // Print out the blocks in the blockchain
        for (int i = 0; i < Blockchain.blockchain.size(); i++) {
            System.out.println("\nBlock " + (i + 1) + " Data: " + Blockchain.blockchain.get(i).hash);
            System.out.println("Previous Hash: " + Blockchain.blockchain.get(i).previousHash);
        }
    }
}
```

OUTPUT:

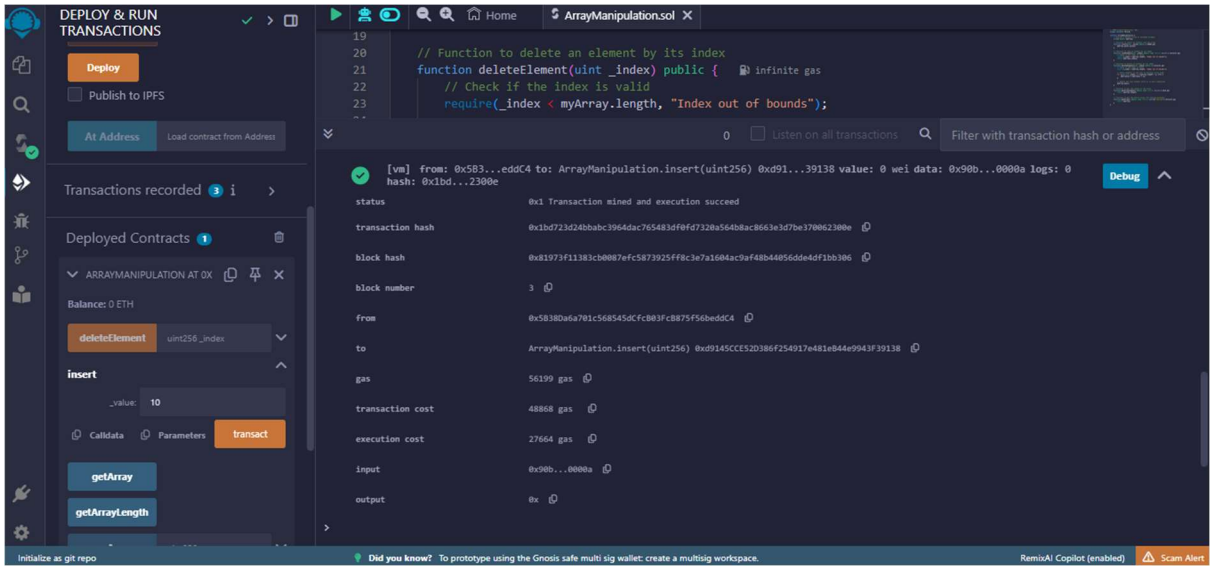
```
Mining Genesis Block...
Block Mined: 000020e1515dd1313f2ca45bbdfa0deeaf503f2068dc7040d75791d8c63db08e
Mining Second Block...
Block Mined: 000004f9a06f625cbed7b93f6ef3ca1518a51b48d12a65f4b0a7a72d4d5a4a52
Mining Third Block...
Block Mined: 0000c9c66af63b84b73ef0660cb5575de759c5d6612eb4755ea570d2534e8645

Blockchain is valid: true

Block 1 Data: 000020e1515dd1313f2ca45bbdfa0deeaf503f2068dc7040d75791d8c63db08e
Previous Hash: 0

Block 2 Data: 000004f9a06f625cbed7b93f6ef3ca1518a51b48d12a65f4b0a7a72d4d5a4a52
Previous Hash: 000020e1515dd1313f2ca45bbdfa0deeaf503f2068dc7040d75791d8c63db08e

Block 3 Data: 0000c9c66af63b84b73ef0660cb5575de759c5d6612eb4755ea570d2534e8645
Previous Hash: 000004f9a06f625cbed7b93f6ef3ca1518a51b48d12a65f4b0a7a72d4d5a4a52
```

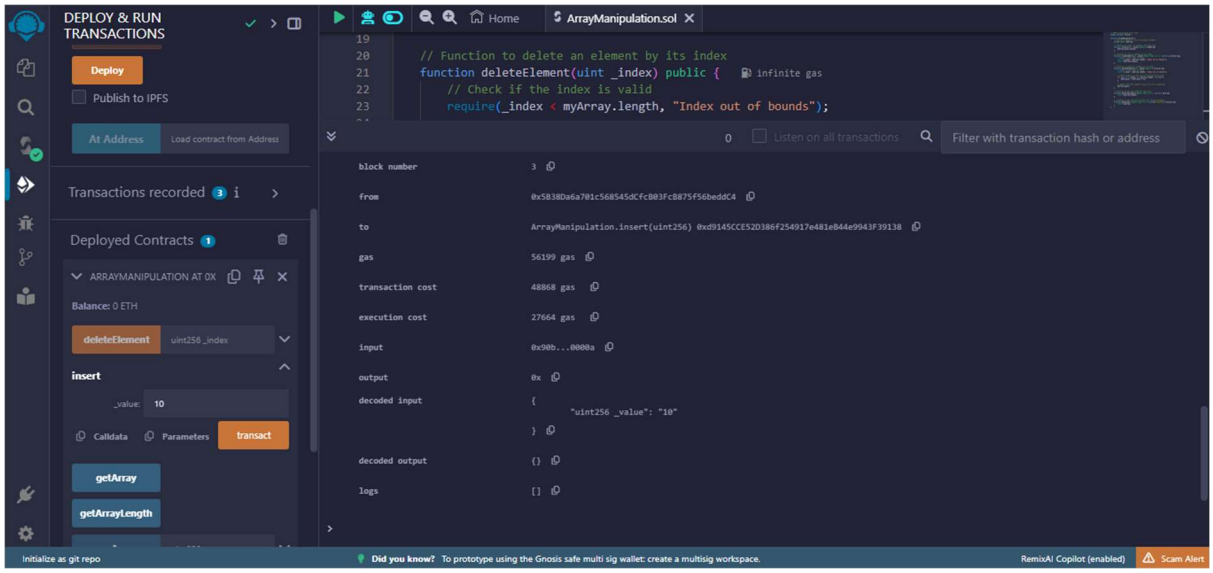
Ex. No: 7 UNDERSTAND THE SOLIDITY VARIABLES AND ARRAYS WITH REGARDS TO FIXED LENGTH ARRAY AND DYNAMIC ARRAY.

PROGRAM:

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

contract ArrayManipulation {
    // Declare a dynamic array of unsigned integers
    uint[] public myArray;

    // Function to insert an element into the array
    function insert(uint _value) public {
        myArray.push(_value);
    }

    // Function to view an element by its index
    function viewElement(uint _index) public view returns (uint) {
        // Check if the index is valid
        require(_index < myArray.length, "Index out of bounds");
        return myArray[_index];
    }

    // Function to delete an element by its index
    function deleteElement(uint _index) public {
        // Check if the index is valid
        require(_index < myArray.length, "Index out of bounds");

        // Shift elements to the left to maintain array order
        for (uint i = _index; i < myArray.length - 1; i++) {
            myArray[i] = myArray[i + 1];
        }

        // Remove the last element since it is now a duplicate
        myArray.pop();
    }

    // Function to get the length of the array
    function getArrayLength() public view returns (uint) {
        return myArray.length;
    }

    // Function to get the entire array (for testing purposes)
    function getArray() public view returns (uint[] memory) {
        return myArray;
    }
}
```

OUTPUT:


DEPLOY & RUN TRANSACTIONS
Deploy
Publish to IPFS
At Address
Load contract from Address
Transactions recorded 3
Deployed Contracts 1
ARRAYMANIPULATION AT 0X
Balance: 0 ETH
deleteElement
uint256 _index
insert
_value: 10
Calldata
Parameters
transact
getArray
getArrayLength
Home
ArrayManipulation.sol
// Function to delete an element by its index
function deleteElement(uint _index) public {
infinite gas
// Check if the index is valid
require(_index < myArray.length, "Index out of bounds");
Listen on all transactions
Filter with transaction hash or address
[vm] from: 0x5B3...eddC4 to: ArrayManipulation.insert(uint256) 0xd91...39138 value: 0 wei data: 0x90b...0000a logs: 0 hash: 0x1bd...2300e
Debug
status 0x1 Transaction mined and execution succeed
transaction hash 0x1bd723d24bbabc3964dac765483df0fd7320a564b8ac8663e3d7be37006230 0e
block hash 0x81973f11383cb0087efc5873925ff8c3e7a1604ac9af48b44056dde4df1bb306
block number 3
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to ArrayManipulation.insert(uint256) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas 56199 gas
transaction cost 48868 gas
execution cost 27664 gas
input 0x90b...0000a
output 0x
Initialize as git repo
Did you know? To prototype using the Gnosis safe multi sig wallet: create a multisig workspace.
RemixAI Copilot (enabled)
Scam Alert

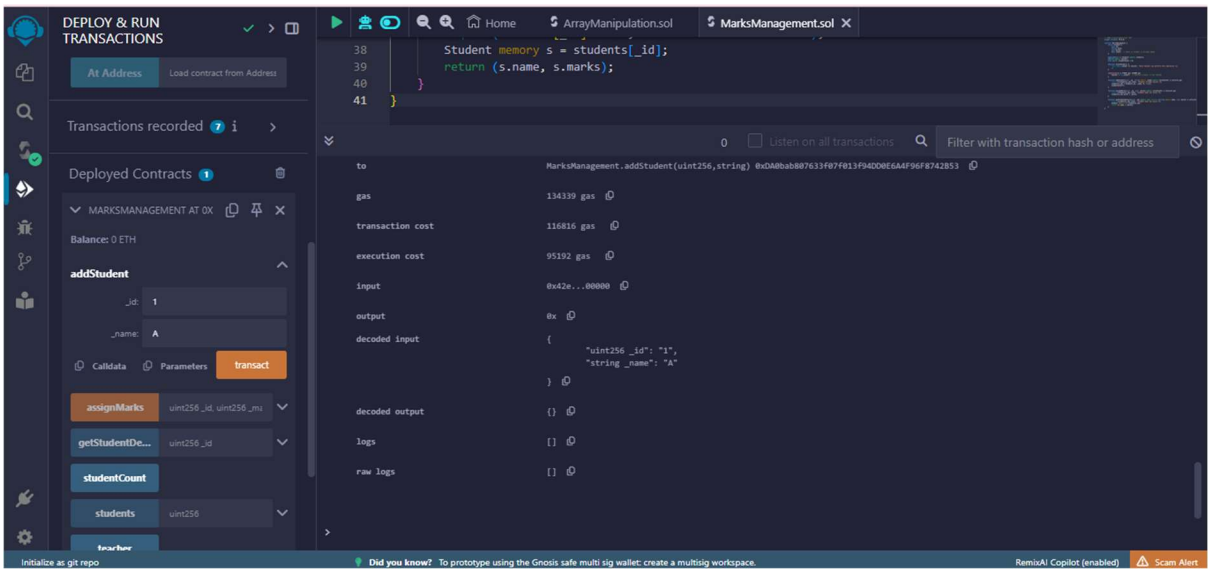

DEPLOY & RUN TRANSACTIONS
Deploy
Publish to IPFS
At Address
Load contract from Address
Transactions recorded 3
Deployed Contracts 1
ARRAYMANIPULATION AT 0X
Balance: 0 ETH
deleteElement
uint256 _index
insert
_value: 10
Calldata
Parameters
transact
getArray
getArrayLength
Home
ArrayManipulation.sol
// Function to delete an element by its index
function deleteElement(uint _index) public {
infinite gas
// Check if the index is valid
require(_index < myArray.length, "Index out of bounds");
Listen on all transactions
Filter with transaction hash or address
block number 3
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to ArrayManipulation.insert(uint256) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas 56199 gas
transaction cost 48868 gas
execution cost 27664 gas
input 0x90b...0000a
output 0x
decoded input { "uint256 _value": "10" }
decoded output {}
logs []
Initialize as git repo
Did you know? To prototype using the Gnosis safe multi sig wallet: create a multisig workspace.
RemixAI Copilot (enabled)
Scam Alert

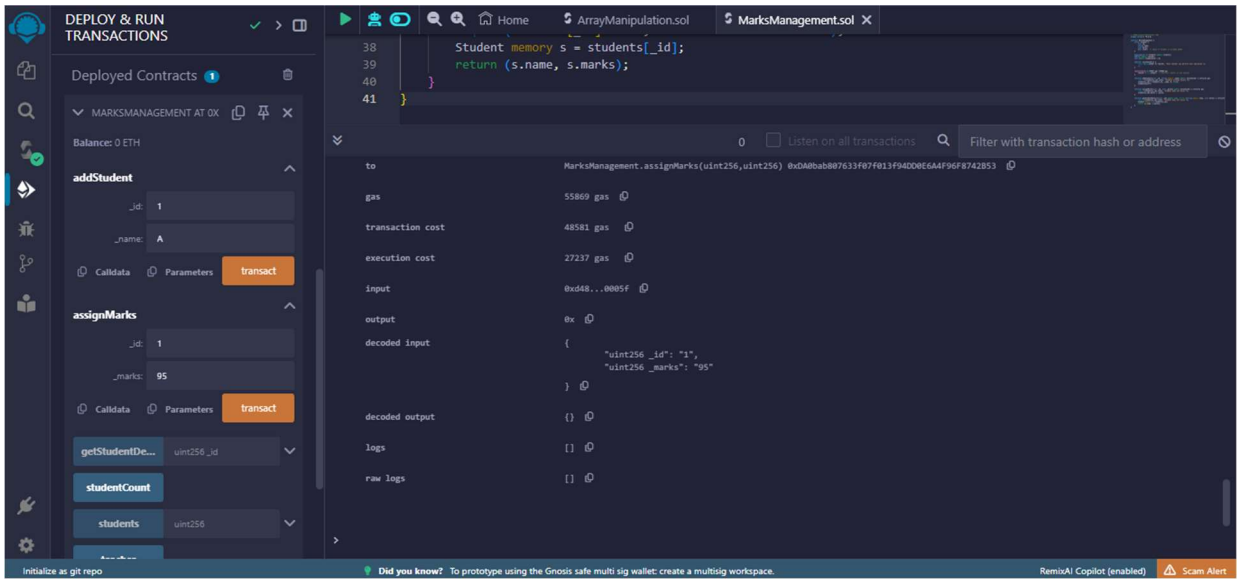
Ex. No: 8 DEPLOY A SMART CONTRACT FOR MARKS MANAGEMENT SYSTEM USING SOLIDITY.

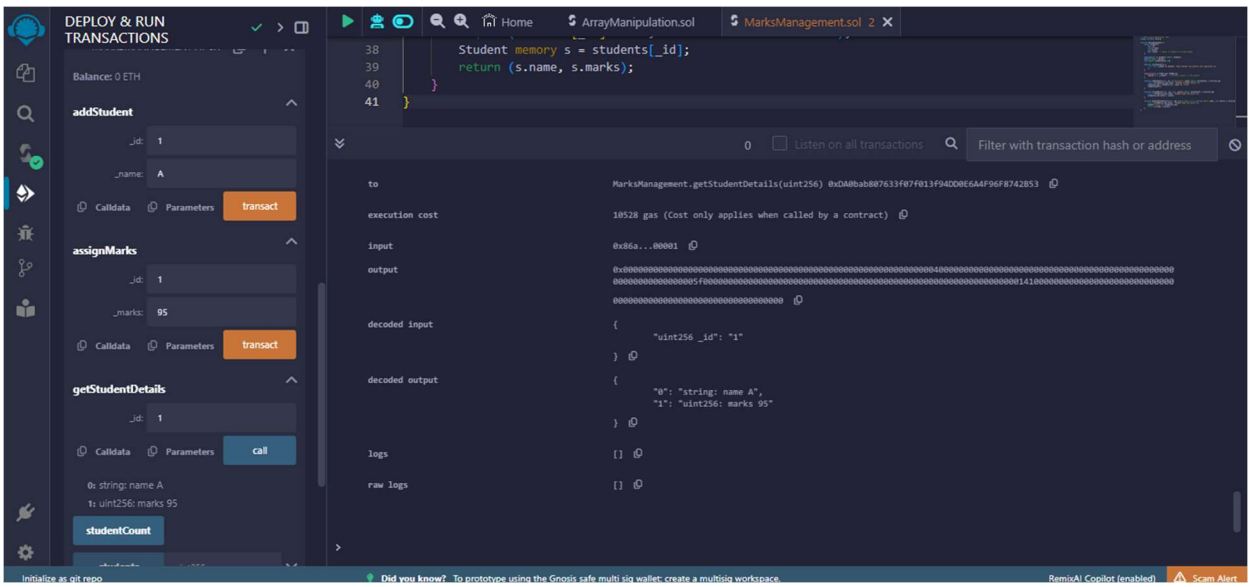
PROGRAM:

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

contract MarksManagement {
    struct Student {
        uint id;
        string name;
        uint marks;
        bool isSet;  // Check if student is already added
    }

    mapping(uint => Student) public students;
    address public teacher;
    uint public studentCount = 0;

    modifier onlyTeacher() {
        require(msg.sender == teacher, "Only teacher can perform this operation.");
        _;
    }

    constructor() {
        teacher = msg.sender;  // Contract creator is the teacher
    }

    function addStudent(uint _id, string memory _name) public onlyTeacher {
        require(!students[_id].isSet, "Student already exists.");
        students[_id] = Student(_id, _name, 0, true);
        studentCount++;
    }

    function assignMarks(uint _id, uint _marks) public onlyTeacher {
        require(students[_id].isSet, "Student does not exist.");
        students[_id].marks = _marks;
    }

    function getStudentDetails(uint _id) public view returns (string memory name, uint marks) {
        require(students[_id].isSet, "Student does not exist.");
        Student memory s = students[_id];
        return (s.name, s.marks);
    }
}
```

OUTPUT:


DEPLOY & RUN TRANSACTIONS
At Address
Load contract from Address
Transactions recorded 7
Deployed Contracts 1
MARKSMANAGEMENT AT 0X
Balance: 0 ETH
addStudent
_id: 1
_name: A
Calldata
Parameters
transact
assignMarks
getStudentDe...
studentCount
students
Home
ArrayManipulation.sol
MarksManagement.sol
Student memory s = students[_id];
return (s.name, s.marks);
Filter with transaction hash or address
to MarksManagement.addStudent(uint256,string) 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
gas 134339 gas
transaction cost 116816 gas
execution cost 95192 gas
input 0x42e...00000
output 0x
decoded input { "uint256 _id": "1", "string _name": "A" }
decoded output {}
logs []
raw logs []
Initialize as git repo
Did you know? To prototype using the Gnosis safe multi sig wallet: create a multisig workspace.
RemixAI Copilot (enabled)
Scam Alert


DEPLOY & RUN TRANSACTIONS
Deployed Contracts 1
MARKSMANAGEMENT AT 0X
Balance: 0 ETH
addStudent
_id: 1
_name: A
Calldata
Parameters
transact
assignMarks
_marks: 95
getStudentDe...
studentCount
students
Home
ArrayManipulation.sol
MarksManagement.sol
Student memory s = students[_id];
return (s.name, s.marks);
Filter with transaction hash or address
to MarksManagement.assignMarks(uint256,uint256) 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
gas 55869 gas
transaction cost 48581 gas
execution cost 27237 gas
input 0xd48...0005f
output 0x
decoded input { "uint256 _id": "1", "uint256 _marks": "95" }
decoded output {}
logs []
raw logs []
Initialize as git repo
Did you know? To prototype using the Gnosis safe multi sig wallet: create a multisig workspace.
RemixAI Copilot (enabled)
Scam Alert


DEPLOY & RUN TRANSACTIONS
Balance: 0 ETH
addStudent
_id: 1
_name: A
Calldata
Parameters
transact
assignMarks
_marks: 95
getStudentDetails
call
0: string: name A
1: uint256: marks 95
studentCount
Home
ArrayManipulation.sol
MarksManagement.sol 2
Student memory s = students[_id];
return (s.name, s.marks);
Filter with transaction hash or address
to MarksManagement.getStudentDetails(uint256) 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
execution cost 10528 gas (Cost only applies when called by a contract)
input 0x86a...00001
output 0x0000000000000000000000000000000000000000000000000000000000000040...
decoded input { "uint256 _id": "1" }
decoded output { "0": "string: name A", "1": "uint256: marks 95" }
logs []
raw logs []
Initialize as git repo
Did you know? To prototype using the Gnosis safe multi sig wallet: create a multisig workspace.
RemixAI Copilot (enabled)
Scam Alert