1. The answer is no. It is not possible to prove within transaction runtime that a particular event with a particular set of arguments has been emitted on the contract in the past. However, events are written to storage, so if a transaction fails after an event is emitted, the event is not reverted.The position of the emit statement inside a function in Solidity does not matter in terms of runtime, but it is recommended to place it at the end of the function.This requires a mechanism that records past events. The 'emit' keyword is used to record events, enabling them to be read by external applications by storing the event on the Ethereum blockchain. Past events can also be queried and desired conditions can be checked using the client called Etherscan.
   You see the code example below.

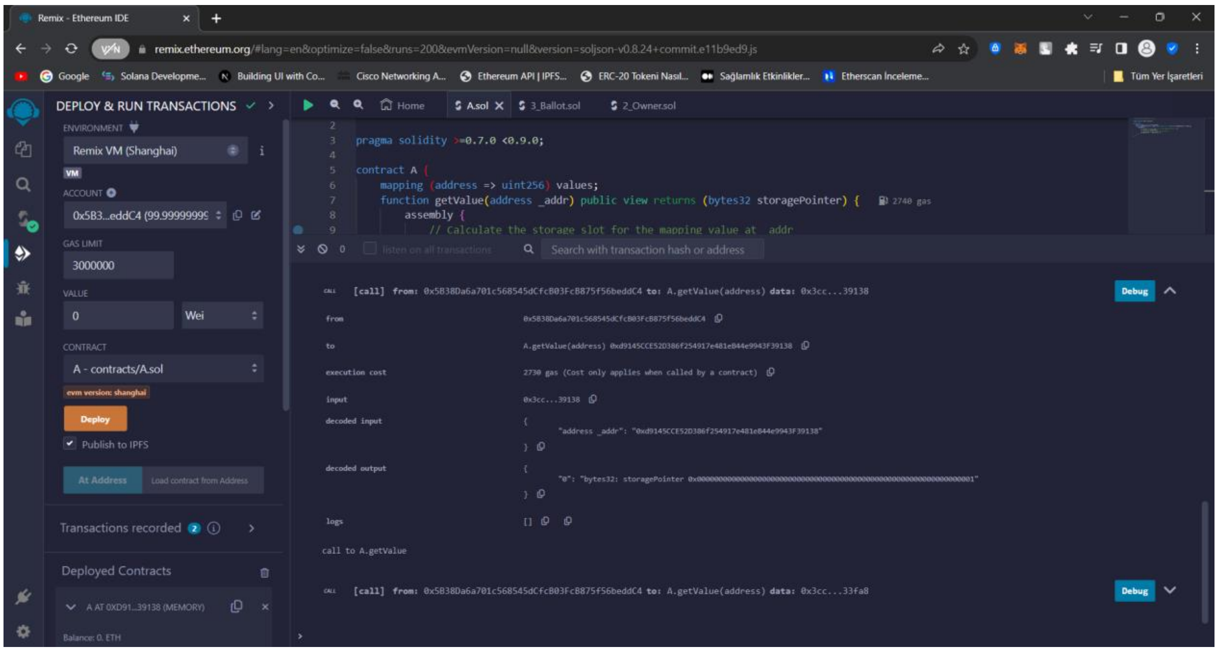
   ```
   event Transfer(address indexed _from, address indexed _to, uint256 _value);

   function transfer(address _to, uint256 _value) public {
       require(_to != address(0));
       require(_value <= balanceOf[msg.sender]);

       balanceOf[msg.sender] -= _value;
       balanceOf[_to] += _value;
       emit Transfer(msg.sender, _to, _value);
   }
   ```

2. This answer is yes. We must first ascertain the precise contract address and the anticipated bytecode in order to confirm if a contract with a given bytecode has been deployed on the Ethereum address. The bytecode of a specific address may then be obtained from the Ethereum blockchain using this information, and it can be compared to the predicted bytecode.
   You can query the code of a specific Ethereum blockchain contract using services like Etherscan.
   Step by step:
   Get the bytecode of the deployed contract at the given address.
   Compare the deployed bytecode with the recompiled bytecode.
   Use source code verification tools.
   Verify the contract on Etherscan.

3. This answer is yes. A solidity contract can call another contract function and the called function can return a struct data structure. Communication between smart contracts: ABI is also used in communication between different smart contracts. In this way, it becomes possible to exchange data and perform transactions between different smart contracts. The 'abi.decode' function can be used to convert the returned data into a data structure of the desired type.
   In other words, it provides direct access to memory using assembly language. Although I do not prefer to use this method, this method can be used (it is a more error-prone process).

Coding Challange 1

```
contract A {
    mapping (address => uint256) values;
    function getValue(address _addr) public view returns (bytes32 storagePointer) {
        assembly {
            // Calculate the storage slot for the mapping value at _addr
            let offset := sload(_addr)
            // The storage pointer should be the offset + 1
            storagePointer := add(offset, 1)
   }
    }
}
```



remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.24+commit.e11b9ed9.js

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Remix VM (Shanghai)

ACCOUNT: 0x5B3...eddC4 (99.99999999

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT: A - contracts/A.sol

evm version: shanghai

Publish to IPFS ✓

Transactions recorded 2

Deployed Contracts

A AT 0XD91...39138 (MEMORY)

Balance: 0. ETH

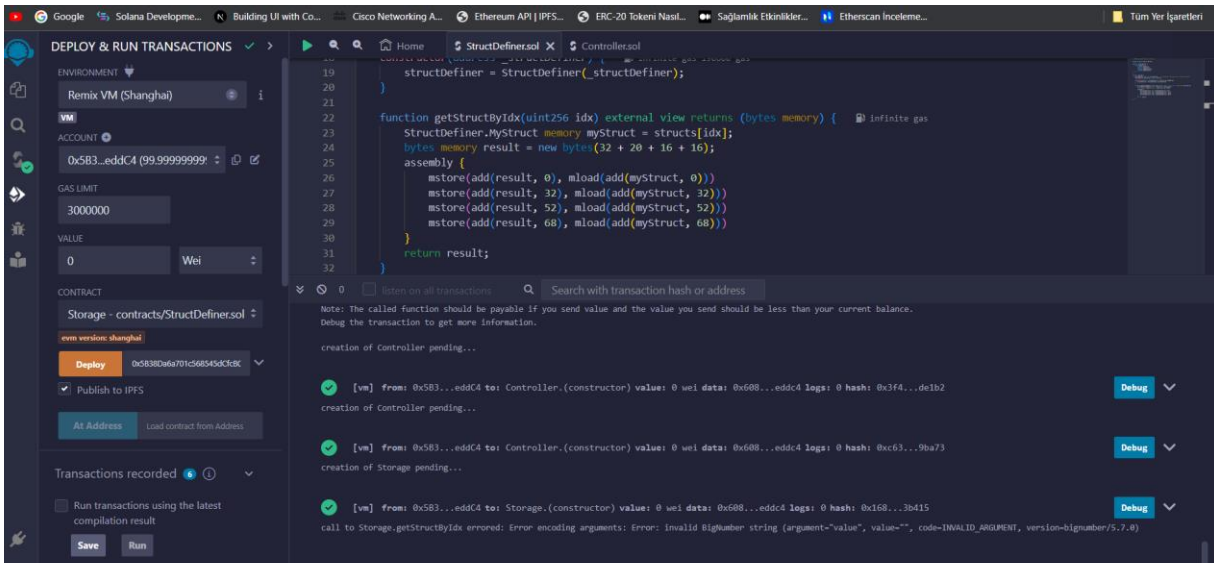
```
pragma solidity >=0.7.0 <0.9.0;

contract A {
    mapping (address => uint256) values;
    function getValue(address _addr) public view returns (bytes32 storagePointer) {    2740 gas
        assembly {
            // Calculate the storage slot for the mapping value at _addr
```

```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: A.getValue(address) data: 0x3cc...39138
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  A.getValue(address) 0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost      2730 gas (Cost only applies when called by a contract)
input               0x3cc...39138
decoded input       {
                        "address _addr": "0xd9145CCE52D386f254917e481eB44e9943F39138"
                    }
decoded output      {
                        "0": "bytes32: storagePointer 0x0000000000000000000000000000000000000000000000000000000000000001"
                    }
logs                []
call to A.getValue
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: A.getValue(address) data: 0x3cc...33fa8
```

Coding Challange 2

```
contract StructDefiner {
  struct MyStruct {
    uint256 someField;
    address someAddress;
    uint128 someOtherField;
    uint128 oneMoreField;
  }
}

contract Storage {
  StructDefiner internal structDefiner; // StructDefiner kontratına bir referans eklendi
  StructDefiner.MyStruct[] internal structs;

  constructor(address _structDefiner) {
    structDefiner = StructDefiner(_structDefiner);
  }

  function getStructByIdx(uint256 idx) external view returns (bytes memory) {
    StructDefiner.MyStruct memory myStruct = structs[idx];
    bytes memory result = new bytes(32 + 20 + 16 + 16);
    assembly {
      mstore(add(result, 0), mload(add(myStruct, 0)))
      mstore(add(result, 32), mload(add(myStruct, 32)))
      mstore(add(result, 52), mload(add(myStruct, 52)))
      mstore(add(result, 68), mload(add(myStruct, 68)))
    }
    return result;
  }
}
```

```
contract Controller {

    Storage internal storageContract;


    constructor(address _storage) {

        storageContract = Storage(_storage);

    }

    function getStruct(uint256 idx) public view returns (uint256 someField, address someAddress,
uint256 someOtherField, uint256 oneMoreField) {

    bytes memory _myStruct = storageContract.getStructByIdx(idx);

    assembly {

        someField := mload(add(_myStruct, 0))

        someAddress := mload(add(_myStruct, 32))

        someOtherField := mload(add(_myStruct, 52))

        oneMoreField := mload(add(_myStruct, 68))

    }

}

}
```


DEPLOY & RUN TRANSACTIONS
Home
StructDefiner.sol
Controller.sol
Deploy
Publish to IPFS
At Address
Load contract from Address
Transactions recorded 6
Run transactions using the latest compilation result
Save
Run
Debug

ENVIRONMENT: Remix VM (Shanghai)
ACCOUNT: 0x5B3...eddC4 (99.99999999
GAS LIMIT: 3000000
VALUE: 0 Wei
CONTRACT: Storage - contracts/StructDefiner.sol
evm version: shanghai
Deploy: 0x5B38Da6a701c568545dCfcB

```
        structDefiner = StructDefiner(_structDefiner);
    }

    function getStructByIdx(uint256 idx) external view returns (bytes memory) {    infinite gas
        StructDefiner.MyStruct memory myStruct = structs[idx];
        bytes memory result = new bytes(32 + 20 + 16 + 16);
        assembly {
            mstore(add(result, 0), mload(add(myStruct, 0)))
            mstore(add(result, 32), mload(add(myStruct, 32)))
            mstore(add(result, 52), mload(add(myStruct, 52)))
            mstore(add(result, 68), mload(add(myStruct, 68)))
        }
        return result;
    }
```

```
Note: The called function should be payable if you send value and the value you send should be less than your current balance.
Debug the transaction to get more information.

creation of Controller pending...

[vm] from: 0x5B3...eddC4 to: Controller.(constructor) value: 0 wei data: 0x608...eddc4 logs: 0 hash: 0x3f4...de1b2
creation of Controller pending...

[vm] from: 0x5B3...eddC4 to: Controller.(constructor) value: 0 wei data: 0x608...eddc4 logs: 0 hash: 0xc63...9ba73
creation of Storage pending...

[vm] from: 0x5B3...eddC4 to: Storage.(constructor) value: 0 wei data: 0x608...eddc4 logs: 0 hash: 0x168...3b415
call to Storage.getStructByIdx errored: Error encoding arguments: Error: invalid BigNumber string (argument="value", value="", code=INVALID_ARGUMENT, version=bignumber/5.7.0)
```