Blockchain Study Notes Day 15:

Module 3 - Solidity Advanced
Chapter 1 - Arrays in Solidity

Introduction to Arrays

Arrays in Solidity are used to store collections of elements of the same data type. They provide a way to handle and manipulate data efficiently within smart contracts.

---

1. Types of Arrays in Solidity

1.1. Fixed-Size Arrays

- Arrays with a predefined size that cannot be changed after declaration.
- **Syntax**:

```
uint[5] public fixedArray;  // Array with 5 elements
```

1.2. Dynamic Arrays

- Arrays without a predefined size that can grow or shrink at runtime.
- **Syntax**:

```
uint[] public dynamicArray;
```

---

2. Array Operations

2.1. Adding Elements to a Dynamic Array

- **Example**:

```
function addElement(uint element) public {
    dynamicArray.push(element);
}
```

2.2. Removing Elements from a Dynamic Array

- **Example**:

```
function removeLastElement() public {
    dynamicArray.pop();
```

```
    }
```

2.3. Accessing Array Elements

- **Example**:

```
function getElement(uint index) public view returns (uint) {
    return dynamicArray[index];
}
```

2.4. Getting Array Length

- **Example**:

```
function getArrayLength() public view returns (uint) {
    return dynamicArray.length;
}
```

---

3. Example Program Demonstrating Arrays (Using Munawar)

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

contract MunawarArrays {
    // Fixed-size array
    uint[3] public fixedArray = [1, 2, 3];

    // Dynamic array
    uint[] public dynamicArray;

    // Function to add an element to the dynamic array
    function addElement(uint element) public {
        dynamicArray.push(element);
    }

    // Function to remove the last element from the dynamic array
    function removeLastElement() public {
        require(dynamicArray.length > 0, "Array is empty.");
        dynamicArray.pop();
    }

    // Function to get an element by index
    function getElement(uint index) public view returns (uint) {
        require(index < dynamicArray.length, "Index out of bounds.");
        return dynamicArray[index];
    }

    // Function to get the length of the dynamic array
    function getDynamicArrayLength() public view returns (uint) {
        return dynamicArray.length;
    }
}
```

---

4. Advanced Array Concepts

4.1. Multi-Dimensional Arrays

- Arrays within arrays, useful for more complex data structures.
- **Example**:

```
uint[][] public multiArray;

function addToMultiArray(uint[] memory newArray) public {
    multiArray.push(newArray);
}
```

4.2. Memory and Storage Arrays

- **Storage Arrays**: Persist data on the blockchain.
- **Memory Arrays**: Temporary arrays used within functions.
- **Example**:

```
function createMemoryArray(uint size) public pure returns (uint[]
memory) {
    uint[] memory tempArray = new uint[](size);
    return tempArray;
}
```

---

5. Best Practices for Using Arrays

- **Bounds Checking**: Always check array bounds to avoid runtime errors.
- **Gas Optimization**: Minimize array operations, especially for large arrays, to save gas.
- **Dynamic vs. Fixed**: Use fixed-size arrays when the size is known to save gas.

Home Task

1. **Modify the Example Program**:
    - Add a function to update an element at a specific index in the dynamic array.
2. **Create a New Contract**:
    - Implement a contract with a multi-dimensional array to manage a 2D data structure.
3. **Research**:
    - Explore gas costs for array operations like `push` and `pop` on dynamic arrays.

Conclusion

Arrays in Solidity are powerful tools for managing collections of data. By understanding their types and operations, developers can effectively manipulate data within smart contracts. However, efficient usage and optimization are crucial to avoid high gas costs and runtime errors.

Day 15 Notes

Prepared by Munawar Johar