**Blockchain Study Notes Day 17:**

**Module 3 - Solidity Advanced**  
**Chapter 3 - Mapping in Solidity**

**Introduction to Mapping**

Mappings in Solidity are used to store key-value pairs, providing a data structure similar to hash tables or dictionaries in other programming languages. They are an essential tool for efficient data retrieval in smart contracts.

**1. What Is Mapping?**

* **Definition**:  
  A mapping is a reference type that associates keys with corresponding values.
* **Purpose**:
  + Efficiently store and retrieve data based on unique keys.
  + Commonly used for maintaining user balances, ownership records, or data lookup tables.

**2. Syntax of Mapping**

**Defining a Mapping**:

mapping(KeyType => ValueType) public mappingName;

* **KeyType**: Any type except for mapping, dynamic arrays, or structs.
* **ValueType**: Can be any type, including mappings or structs.

**3. Example of Mapping (Using Munawar)**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract MunawarMapping {

// Mapping to store balances

mapping(address => uint) public balances;

// Function to set balance for a user

function setBalance(address \_user, uint \_amount) public {

balances[\_user] = \_amount;

}

// Function to get balance of a user

function getBalance(address \_user) public view returns (uint) {

return balances[\_user];

}

// Function to reset balance of a user

function resetBalance(address \_user) public {

delete balances[\_user];

}

}

**Explanation**:

1. **balances**: A mapping from addresses to their corresponding balances.
2. **setBalance**: Updates the balance for a specific user.
3. **getBalance**: Retrieves the balance of a specific user.
4. **resetBalance**: Deletes the balance entry for a user.

**4. Operations on Mappings**

**4.1. Setting Values**

* Assign a value to a key:

balances[msg.sender] = 100;

**4.2. Accessing Values**

* Retrieve the value associated with a key:

uint balance = balances[msg.sender];

**4.3. Deleting Values**

* Remove the mapping entry for a key:

delete balances[msg.sender];

**5. Advanced Mapping Concepts**

**5.1. Nested Mappings**

* Mappings within mappings for more complex relationships.
* **Example**:

mapping(address => mapping(uint => uint)) public nestedMapping;

function setNestedValue(address \_user, uint \_id, uint \_value) public {

nestedMapping[\_user][\_id] = \_value;

}

function getNestedValue(address \_user, uint \_id) public view returns (uint) {

return nestedMapping[\_user][\_id];

}

**5.2. Mapping with Structs**

* Use structs as the value type for more complex data storage.
* **Example**:

struct User {

string name;

uint balance;

}

mapping(address => User) public users;

function createUser(address \_user, string memory \_name, uint \_balance) public {

users[\_user] = User(\_name, \_balance);

}

**6. Best Practices for Using Mappings**

* **Efficient Lookups**:
  + Use mappings for scenarios requiring fast and efficient lookups.
* **Avoid Iteration**:
  + Mappings do not support iteration over keys. Use events or external tools to track keys if necessary.
* **Gas Optimization**:
  + Mappings are efficient for large datasets but avoid unnecessary updates to minimize gas usage.

**Home Task**

1. **Extend the Example Program**:
   * Add a function to transfer balance between two addresses.
2. **Create a New Contract**:
   * Implement a contract to track product ownership using a mapping of product IDs to owner addresses.
3. **Research**:
   * Explore real-world examples of mapping usage in decentralized finance (DeFi) contracts.

**Conclusion**

Mappings in Solidity are a powerful tool for storing and retrieving data efficiently. By understanding their capabilities and limitations, developers can design robust and scalable smart contracts for a wide range of applications.

.

Day 17 Notes

***Prepared by Munawar Johar***