Mapping and Structs

Table of Content

Solidity Mapping and Structs

Explanation of Mapping and Structs in Solidity:

Mapping:

Mappings in Solidity are key-value stores similar to dictionaries or hash maps in other programming languages. They associate unique keys with values. In Solidity, mappings are commonly used to create collections of data where each piece of data is associated with a unique identifier.

Real-life Example: Imagine a library where each book has a unique ID. A mapping can be used to associate each book ID with its details like title, author, and genre.

Syntax: mapping (uint256 => Book) public books;

Structs:

Structs in Solidity are user-defined data structures that allow grouping multiple variables under a single name. They enable the creation of more complex data types to represent a collection of related variables.

Real-life Example: Consider a car manufacturing company storing information about cars, such as model, make, and year of manufacture, as a single entity.

Syntax: struct Car {string model; string make; uint256 year; }

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

contract MappingStructDemo {
    // Defining a struct for a Product
    struct Product {
        uint256 productId;
        string productName;
        uint256 price;
    }

    // Mapping to associate Product IDs with Product struct
    mapping(uint256 => Product) public products;

    // Function to add product details using the struct
     function addProduct(uint256 _productId, string memory _productName, uint256 _price)
public {
```

```
        Product memory newProduct = Product(_productId, _productName, _price);
        products[_productId] = newProduct;
    }

    // Function to retrieve product details using the product ID
     function getProduct(uint256 _productId) public view returns (uint256, string memory,
uint256) {
        Product memory retrievedProduct = products[_productId];
                        return      (retrievedProduct.productId,      retrievedProduct.productName,
retrievedProduct.price);
    }
}
```

Code Explanation:

- **Struct Product:**
    - Represents a product with three attributes: **productId**, **productName**, and **price**.
- **Mapping products:**
    - Associates each **productId** (key) with its corresponding **Product** struct (value).
- **Function addProduct:**
    - Adds a new product by creating a **Product** struct and assigning it to the **products** mapping using the provided **_productId** as the key.
- **Function getProduct:**
    - Retrieves product details by providing the **_productId**.
    - Uses the **_productId** as a key to fetch the associated **Product** struct from the **products** mapping and returns its attributes.

Imagine an e-commerce platform managing product information. The **Product** struct represents items sold on the platform, and the **products** mapping stores these items with unique IDs for efficient retrieval.

This smart contract allows adding new products and retrieving product details by their IDs, leveraging the use of structs and mappings to organize and access product data efficiently.

Mapping, Structs, and Error Handling in Solidity help organize data efficiently and manage unexpected situations within smart contracts.

Mapping:

Mappings are like dictionaries or address books, associating keys with values. They provide a way to store and retrieve data using a unique identifier. For instance, in a token contract, a mapping can link addresses to their token balances.

Structs:

Structs are blueprints for creating custom data structures. They allow grouping multiple variables together. Think of them as creating a new 'type' that holds various pieces of information. For instance, in an auction contract, a struct could define properties like item details or bidder information.