Solidity Hack 101 - level 2

Introduction To Array

> An array allows us to represent a collection of data, but it is often more useful to think of an array as a collection of variables of the same type with keys having sequential order based off the order they were added.

Instead of declaring individual variables, such as number1, number2, number3, ….., and number99, you can declare one array variable such as *numbers* and use numbers[0], numbers[1], …, numbers[99] to represent individual variables.

In Solidity, There is two types of arrays in solidity, which are the *storage array* and the ==_memory array.

Storage Array:

These are arrays that are stored inside the blockchain after you execute your function, you can declare them like normal variables without our contract like below:

```
pragma solidity ^0.5.3;

contract Array {
||uint[] intergerArray; //sample showing initialization of an array of integers|
|||
||bool[] boolArray; //sample showing initialization of an array of booleans|
|||
||address[] addressArray; //sample showing initialization of an array of address etc.|
||}|
```

where *unit/bool/address* is the variable type, followed by *[]* and then the name of the array which can be any name.

Manipulation of the storage array:

To add a new element to the array, we need to reference the array and use the .push(value), we can also update an array position by referencing the element key like `myArray[0] = ‘new value’;`, we also can also get a particular array element position by simple using `myArray[0]` where `0` is the index or key of the array element

```
pragma solidity ^0.5.3;



contract{

   uint[] myArray;

   function manipulateArray() external {
    myArray.push(1); // add an element to the array
    myArray.push(3); // add another element to the array
    myArray[0]; // get the element at key 0 or first element in the array
```

```
     myArray[0] = 20; // update the first element in the array

    //we can also get the element in the array using the for loop

    for (uint j = 0; j < myArray.length; j++) {
       myArray[j];
       }
    }
}
```

We can also use the for loop to access the element of the array as we would array in javascript (also in the snippet above). To delete an element in the array, we will use `delete myArray[1]` where 1 is the array key, please note that this will not affect the array length but will only reset the array value at the index of one or second element in the array to the default value of the data type. i.e 0 if it's a uint type and false if bool. When we try to access an array key that does not exist, we will have an error.

Memory Array:

These arrays are not stored into the blockchain after calling the function, you can only declare them within a function like below:

```
pragma solidity ^0.5.3;

contract MemoryArray {
/**
* These are temporary arrays and only exists when you are executing a function|
* Must have a fixed size and must be declared inside a function|
* You cannot run a for loop in a memory array|
*/
function memoryArray() public {
   uint[] memory newArray = new uint[](10);
  }
}
```

where `unit[]` is the type of the variables, `memory` keyword declares it has a memory array followed by the name which can be an acceptable name, it must be equated to `new unit[](10)` type where the `unit[]` must be the variable type again and the number of elements that will be in the array in the bracket `()`.

Manipulation of the memory array:

To add value to the memory array, we cannot use the `.push()` method, as we need to use the index notation instead like `newArray[0] = 1` , `newArray[1] = 1` etc. We can read a value, update a value, delete a value from the memory array just like the storage array.

Passing an array as a function argument:

We can pass an array as a function argument in solidity, the function visibility determines if the keyword before the array name should be `calldata` or `memory`. We use `calldata` if the function visibility is `external` and `memory` if the function visibility is public and internal. See example below:

```
pragma solidity ^0.5.3;

contract ArraySolidity {

/** Visible external */
function functionExternal(uint[] calldata myArg) external returns(uint[] memory) {

    uint[] memory newArray = new uint[](10);
    newArray[0] = 1;
    return newArray;
}
```

```
/** Visible public */
function functionPublic(uint[] memory myArg) public returns(uint[] memory) {
   uint[] memory newArray = new uint[](10);
   newArray[0] = 1;
   return newArray;
  }
}
```

Mapping

Mapping in Solidity acts like a hash table or dictionary in any other language. These are used to store the data in the form of key-value pairs, a key can be any of the built-in data types but reference types are not allowed while the value can be of any type.

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

contract Counter {
    // Mapping
    mapping(uint => string) names;
    mapping(uint => book) public books;

    struct book{
        string title;
        string author;
    }

    constructor() public {
        names[1] = "Adam";
        names[2] = "Bruce";
        names[3] = "Carl";
    }

    function addbook(
        uint _id,
        string memory _title,
        string memory _author
    ) public {
        books[_id] = book(_title,_author);

    }
}
```