**PRACTICAL NO.: 3A (continue)**

**AIM: WRITE A SOLIDITY PROGRAM FOR STRING, ARRAYS, ENUMS, STRUCTURE & MAPPINGS.**

**A) String:**

Solidity supports String literal using both double quote (") and single quote ('). It provides string as a data type to declare a variable of type String.(Int to str)

pragma solidity ^0.5.0;

contract SolidityTest {

constructor() public{

}

function getResult() public view returns(string memory){

uint a = 1;

uint b = 2;

uint result = a + b;

return integerToString(result);

}

function integerToString(uint \_i) internal pure

returns (string memory) {

if (\_i == 0) {

return "0";

}

uint j = \_i;

uint len;

while (j != 0) {

len++;

j /= 10;

}

bytes memory bstr = new bytes(len);

uint k = len - 1;

while (\_i != 0) {

bstr[k--] = byte(uint8(48 + \_i % 10));

\_i /= 10;

}

return string(bstr);

}

}

**B)Array:**

Array is a data structure, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

// Solidity program to demonstrate

// accessing elements of an array

pragma solidity ^0.5.0;

// Creating a contract

contract Types {

// Declaring an array

uint[6] data;

uint x;

// Defining function to

// assign values to array

function array\_example() public returns (uint[6] memory)

{

data  = [uint(10), 20, 30, 40, 50, 60];

}

function result() public view returns(uint[6] memory){

return data;

}

// Defining function to access

// values from the array

// from a specific index

function array\_element() public view returns (uint){

uint x = data[2];

return x;

}

}

**C)Enums:**

Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums. With the use of enums it is possible to reduce the number of bugs in your code.

// Solidity program to demonstrate

// how to use 'enumerator'

pragma solidity ^0.5.0;

// Creating a contract

contract Types {

// Creating an enumerator

enum week\_days

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

// Declaring variables of

// type enumerator

week\_days week;

week\_days choice;

// Setting a default value

week\_days constant default\_value

= week\_days.Sunday;

// Defining a function to

// set value of choice

function set\_value() public {

choice = week\_days.Thursday;

}

// Defining a function to

// return value of choice

function get\_choice(

) public view returns (week\_days) {

return choice;

}

// Defining function to

// return default value

function getdefaultvalue(

) public pure returns(week\_days) {

return default\_value;

}

}

**D)Structure:**

Struct types are used to represent a record.

pragma solidity ^0.5.0;

contract test {

struct Book {

string title;

string author;

uint book\_id;

}

Book book;

function setBook() public {

book = Book('Learn Java', 'TP', 1);

}

function getBookId() public view returns (uint) {

return book.book\_id;

}

}

**E)Mappings:**

Mapping is a reference type as arrays and structs. Following is the syntax to declare a mapping type.

mapping(\_KeyType => \_ValueType) where ,

**\_KeyType** − can be any built-in types plus bytes and string. No reference type or complex objects are allowed.

**\_ValueType** − can be any type.

pragma solidity ^0.5.0;

contract LedgerBalance {

mapping(address => uint)  balance;

function updateBalance() public  returns(uint) {

balance[msg.sender]=30;

return balance[msg.sender];

}

}

**Mapping program for String.**

pragma solidity ^0.5.0;

contract LedgerBalance {

mapping(address => string)  name;

function updateBalance() public returns(string memory){

name[msg.sender] = "Mrunali";

return name[msg.sender];

}

function printsender() public view returns(address) {

return msg.sender;

}

}