**PRACTICAL NO.:4**

**AIM: WRITE A SOLIDITY PROGRAM FOR CONTRACT, INHERITANCE, CONSTRUCTORS, ABSTRACT CONTRACTS, INTERFACES, LIBRARIES, ASSEMBLY, EVENTS, ERROR HANDLING.**

**A)Contract:**

Contract in Solidity is similar to a Class in C++. A Contract have following properties.

**Constructor** − A special function declared with constructor keyword which will be executed once per contract and is invoked when a contract is created.

**State Variables** − Variables per Contract to store the state of the contract.

**Functions** − Functions per Contract which can modify the state variables to alter the state of a contract.

// Calling function from external contract

pragma solidity ^0.5.0;

contract C {

//private state variable

uint private data;

//public state variable

uint public info;

//constructor

constructor() public {

info = 10;

}

//private function

function increment(uint a) private pure returns(uint) { return a + 1; }

//public function

function updateData(uint a) public { data = a; }

function getData() public view returns(uint) { return data; }

function compute(uint a, uint b) internal pure returns (uint) { return a + b; }

}

//Derived Contract

contract E is C {

uint private result;

C private c;

constructor() public {

c = new C();

}

function getComputedResult() public {

result = compute(3, 5);

}

function getResult() public view returns(uint) { return result; }

function getData() public view returns(uint) { return c.info(); }

}

**B)Inheritance:**

Inheritance is a way to extend functionality of a contract. Solidity supports both single as well as multiple inheritance.

// Solidity program to

// demonstrate

// Single Inheritance

pragma solidity >=0.4.22 <0.6.0;

// Defining contract

contract parent{

// Declaring internal

// state variable

uint internal sum;

// Defining external function

// to set value of internal

// state variable sum

function setValue() external {

uint a = 20;

uint b = 20;

sum = a + b;

}

}

// Defining child contract

contract child is parent{

// Defining external function

// to return value of

// internal state variable sum

function getValue() external view returns(uint) {

return sum;

}

}

// Defining calling contract

contract caller {

// Creating child contract object

child cc = new child();

// Defining function to call

// setValue and getValue functions

function testInheritance() public {

cc.setValue();

}

function result() public view returns(uint ){

return cc.getValue();

}

}

**C)Constructors:**

Constructor is a special function declared using constructor keyword. It is an optional function and is used to initialize state variables of a contract. Following are the key characteristics of a constructor.

A contract can have only one constructor.

A constructor code is executed once when a contract is created and it is used to initialize contract state.

A constructor can be either public or internal.

An internal constructor marks the contract as abstract.

In case, no constructor is defined, a default constructor is present in the contract.

pragma solidity ^0.5.0;

contract Base {

uint data;

constructor(uint \_data) public {

data = \_data;

}

function getresult()public view returns(uint){

return data;

}

}

contract Derived is Base (5) {

constructor() public {}

}

**// Indirect Initialization of Base Constructor**

pragma solidity ^0.5.0;

contract Base {

uint data;

constructor(uint \_data) public {

data = \_data;

}

function getresult()public view returns(uint){

return data;

}

}

contract Derived is Base {

constructor(uint \_info) Base(\_info \* \_info) public {}

}

**D)Abstract Contracts:**

Abstract Contract is one which contains at least one function without any implementation. Such a contract is used as a base contract. Generally an abstract contract contains both implemented as well as abstract functions. Derived contract will implement the abstract function and use the existing functions as and when required.

pragma solidity ^0.5.0;

contract Calculator {

function getResult() public view returns(uint);

}

contract Test is Calculator {

function getResult() public view returns(uint) {

uint a = 4;

uint b = 2;

uint result = a + b;

return result;

}

}

**E)Interfaces:**

Interfaces are similar to abstract contracts and are created using interface keyword. Following are the key characteristics of an interface.

Interface can not have any function with implementation.

Functions of an interface can be only of type external.

Interface can not have constructor.

Interface can not have state variables.

pragma solidity ^0.5.0;

interface Calculator {

function getResult() external view returns(uint);

}

contract Test is Calculator {

constructor() public {}

function getResult() external view returns(uint){

uint a = 5;

uint b = 2;

uint result = a + b;

return result;

}

}