Introduction to Solidity

M. Maddah-Ali

Sharif Blockchain Lab
Sharif University of Technology
Electrical Engineering Department

Table of contents

Contract

Tips:

- Similar to classes in object-oriented languages
- Contain persistent data in state variables
- Functions that can modify variables

Structure

Overview

```
pragma solidity Version;
contract Name {
        // State Variables

        // Modifiers

        // Events

        // Constructor

        // Other functions
}
```

- Source files should be annotated with a version pragma to reject being compiled with future compiler versions that might introduce incompatible changes
- Releases that contain breaking changes will always have versions of the form 0.X.0 or X.0.0

Storage

```
pragma solidity Version;
contract Name {
        //State Variables
. . .
}
```

Storages of the contract which the state saves them

Example

A simple contract

```
pragma solidity ^0.4.0;
contract SimpleStorage {
        uint storedData;
}
```

Example

A simple contract

```
pragma solidity ^0.4.0;
contract SimpleStorage {
        uint public storedData;
}
```

The user can read the value of variable directly by code without any function

Although, a variable which is not public can be read on Blockchain!

Variable Types

Booleans

- **bool: The possible values are constants true and false**
- **Operators:**
  - ! (logical negation)
  - && (logical conjunction, “and”)
  - || (logical disjunction, “or”)
  - == (equality)
  - != (inequality)
- **Example:**
  - **bool x = true;**

Variable Types

Integers

- int / uint: Signed and unsigned integers of various sizes.
- Keywords uint8 to uint256 in steps of 8 (unsigned of 8 up to 256 bits) and int8 to int256.
- uint and int are aliases for uint256and int256, respectively.
- Operators:
  - Comparisons: <=, <, ==, !=, >=, > (evaluate to bool)
  - Bit operators: &, |, ^, ~
  - Arithmetic operators: +, -, *, /, %, **, <<, >>
- Example:
  - int x = 1;

Variable Types

Address

- Holds a 20 byte value.
- **Operators:**
  - <=, <, ==, !=, >= and >
- **Example:**
  - address my_address = 0x123;

Mostly we do not set the value of an address manually

More Info: https://solidity.readthedocs.io/en/v0.5.1/types.html#members-of-addresses

Variable Types

Fixed-size byte arrays

- bytes1, bytes2, bytes3, …, bytes32. byte is an alias for bytes1.
- **Example:**
  - byte x = 1;

Variable Types

Dynamically-sized byte arrays

- bytes
  - Dynamically-sized byte array not a value-type!
  - **Example:**
    - bytes x = 0x123;
- String
  - Dynamically-sized UTF-8-encoded string not a value-type!
  - **Example:**
    - string x = ‘solidity’;

Variable Types

Enums

- A categorical variable which has multi states
- **Example:**
  - enum actions {Left, Right, Up, Down}

Variable Types

Arrays

- **Fixed Size**
  - An array of fixed size k and element type T is written as T[](k)
- **Dynamic Size**
  - An array of dynamic size and element type T is written as T[ ]
  - **Example:**
    - uint[] x = new uint[](7)
    - uint[] x

Variable Types

Structs

- A composite data type allowing the different variables to be accessed via a single pointer
- **Example:**

```
struct Campaign {
        address beneficiary;
        uint numFunders;
        uint amount;
        string name;
}
```

Variable Types

Mappings

- Mapping types are declared as:
  - mapping(_KeyType =>_ValueType)
- Mappings can be seen as hash tables
- We can't access the list of all keys or values of a mapping
- **Example:**
  - mapping(address => uint) balances;

Function

Function Types

function name (<parameter types>) {1} [2] [**returns** (<return types>)]

- 1: **visibility**
- 2: **pure or view (optional)**

Function

Visibility

- External
  - External functions are part of the contract interface, which means they can be called from other contracts and via transactions. An external function f cannot be called internally (i.e. f()does not work, but this.f() works). External functions are sometimes more efficient when they receive large arrays of data.
- Public
  - Public functions are part of the contract interface and can be either called internally or via messages. For public state variables, an automatic getter function (see below) is generated.
- Internal
  - Those functions and state variables can only be accessed internally (i.e. from within the current contract or contracts deriving from it), without using this.
- Private
  - Private functions and state variables are only visible for the contract they are defined in and not in derived contracts.

https://solidity.readthedocs.io/en/v0.5.1/contracts.html#visibility-and-getters

Function

Pure and View

- pure
  - Functions can be declared pure in which case they promise not to read from or modify the state
- view
  - Functions can be declared view in which case they promise not to modify the state.

https://solidity.readthedocs.io/en/v0.5.1/contracts.html#pure-functions

https://solidity.readthedocs.io/en/v0.5.1/contracts.html#view-functions

Example

- Let complete our simple contract

```
pragma solidity ^0.4.0;
contract SimpleStorage {
        uint storedData;
}
```

Example

- Let complete our simple contract

```
pragma solidity ^0.4.0;
contract SimpleStorage {
        uint storedData;
        function set(uint x) public {
                storedData = x;
        }
        function get() public view returns (uint) {
                return storedData;
        }
}
```

Local vs State Variables

```
pragma solidity ^0.4.0;
contract my_contract{
        uint state_variable;
        // same as global storage variable
        function test() returns uint{
                uint local_variable = 10;
                 // same as local storage variable
                return local_variable * state_variable;
        }
}
```

These variables are the only variable which the network consider them as storage and save them

These variables are temporary and the network does not save them

Constructor

- Constructor is a function
- Only one constructor
- Only is called after deploying
- Initialization of variables
- Should be public

Example

```
pragma solidity ^0.4.0;
contract SimpleStorage {
        uint storedData;
        function set(uint x) public {
                storedData = x;
        }
        function get() public constant returns (uint) {
                return storedData;
        }
}
```

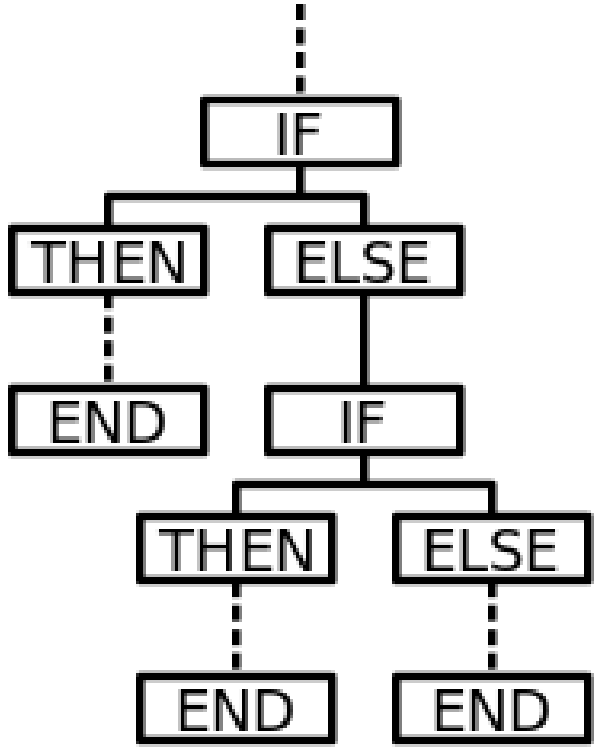
Example

```
pragma solidity ^0.4.0;
contract SimpleStorage {
        uint storedData;
        constructor(uint x) public {
                storedData = x;
        }
        function set(uint x) public {
                storedData = x;
        }
        function get() public constant returns (uint) {
                return storedData;
        }
}
```

Control Structures

- If

```
uint bar = 5;
if (true) {
        i += j;
}
else {
        uint j = 10;// never executes
}
```



Control Structures

For

```
for (uint i = 0; i < 10; i++) {
        //instructions that can be related to i or not
}
```

Control Structures

While

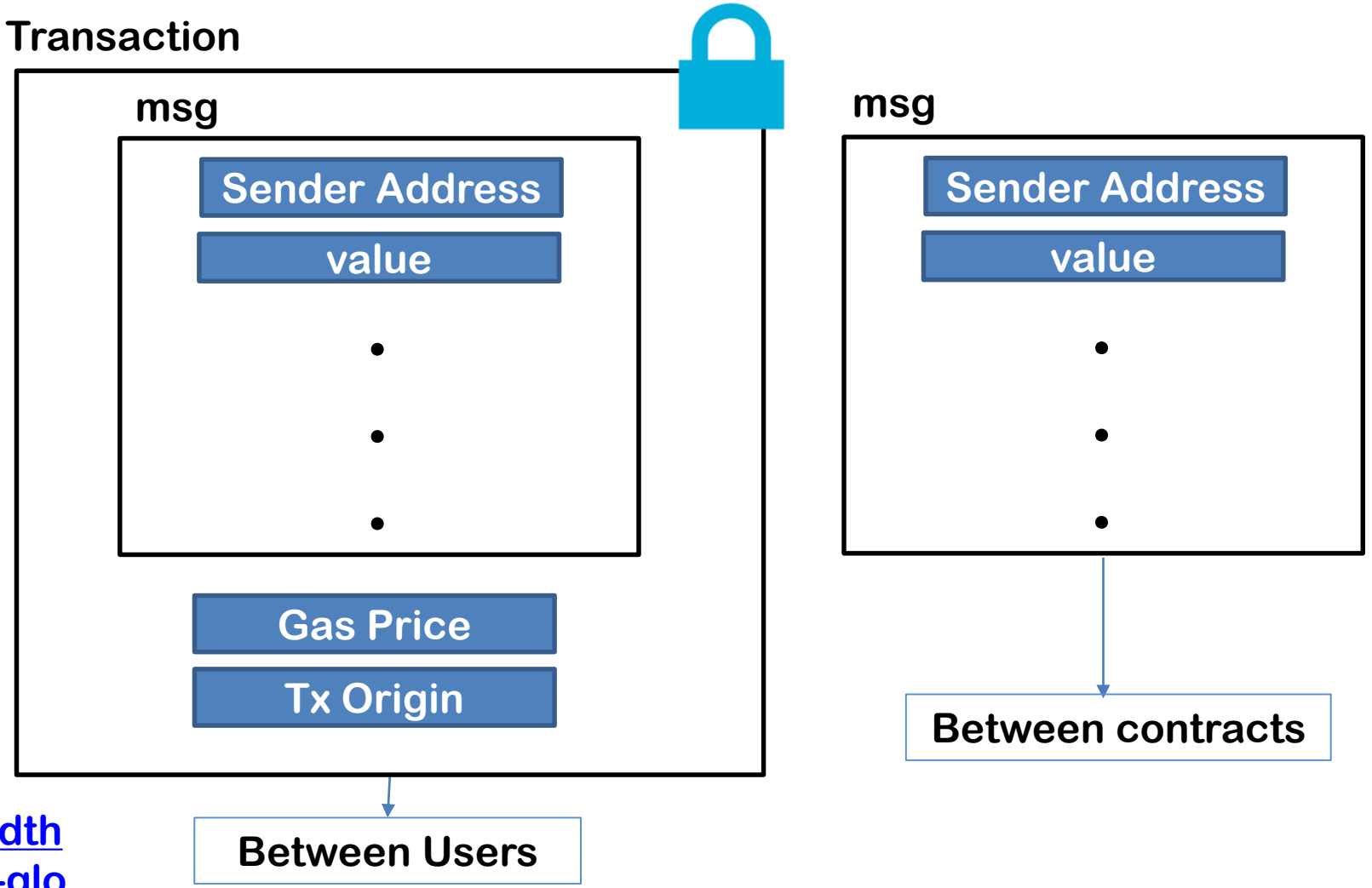
```
uint i = 0;
while (i++ < 10) {
        //instructions that can be related to i or not
}
```

Payable

- Payable keyboard can be used for functions and addresses
- It means that you can send ethers to that address or call that function with value set.
- **Notice** that if you don't use this keyboard you **can't** send ethers to that address or function.
- Example
  - address payable owner;
  - function payToMe() public payable { // Do something }

Message

- msg



More Info: https://solidity.readthedocs.io/en/v0.5.1/units-and-global-variables.html - block-and-transaction-properties

Message

- msg

```
pragma solidity ^0.4.0;
contract mycontract {
    address chairperson;
    function mycontract() public {
        chairperson = msg.sender;
    }
}
```

Require

```
pragma solidity ^0.4.0;
contract SimpleContract{
        address owner;
        uint storedData;
        constructor() public{
                owner = msg.sender;
        }
        function set(uint x) public {
                if(msg.sender == owner){
                        storedData = x;
                }
        }
}
```

Require

```
pragma solidity ^0.4.0;
contract SimpleContract{
        address owner;
        uint storedData;
        constructor() public{
                owner = msg.sender;
        }
        function set(uint x) public {
                // if condition is not met all
                // state changes will be undone
                require(msg.sender == owner);
                storedData = x;
        }
}
```

More Info: https://solidity.readthedocs.io/en/v0.5.1/control-structures.html#error-handling-assert-require-revert-and-exceptions

Modifier

```
pragma solidity ^0.4.0;
contract SimpleContract{
        address owner;
        modifier onlyOwner() {
                require(msg.sender == owner);
                // Do not forget the "_;"! It will be replaced by
                // the actual function body when the modifier is used.
                _;
        }
        uint storedData;
        constructor() public{
                owner = msg.sender;
        }
        function set(uint x) onlyOwner()  public {
                storedData = x;
        }
}
```

More Info: https://solidity.readthedocs.io/en/v0.5.1/common-patterns.html#restricting-access

Inheritance

```
contract owned {
        constructor() public { owner = msg.sender; }
        address payable owner;
}

// Use `is` to derive from another contract. Derived
// contracts can access all non-private members including
// internal functions and state variables.
contract mortal is owned {
        function kill() public {
                if (msg.sender == owner) selfdestruct(owner);
        }
}
```

More Info: https://solidity.readthedocs.io/en/v0.5.1/contracts.html#inheritance

Inheritance

Override

```
contract RestWhenRich is Mortal {
        string[] public payers;
        function earnMoney(string payerName) payable public {
                payers.push(payerName);
        }
        // Override kill function of Mortal contract
        function kill() public {
                if (address(this).balance >= 1000 ether && msg.sender == owner)
                        selfdestruct(owner);
        }
}
```

More Info: https://solidity.readthedocs.io/en/v0.5.1/contracts.html#inheritance

Message to Another Account

address.send()

Account type can be Externally Controlled or Contract

```
pragma solidity ^0.4.0;
contract SimpleContract{
        address payable owner;
        constructor() payable public{
                owner = msg.sender;
        }
        function sendEther(address payable receiverAddress) external{
                // send returns false on failure
                // forwards 2,300 gas stipend (not adjustable), safe against reentrancy
                if ( !receiverAddress.send(1 ether))
                        revert();
        }
}
```

More Info: https://solidity.readthedocs.io/en/v0.5.1/units-and-global-variables.html#members-of-address-types

Message to Another Account

address.transfer()

Account type can be Externally Controlled or Contract

```
pragma solidity ^0.4.0;
contract SimpleContract{
        address payable owner;
        constructor() payable  public{
                owner = msg.sender;
        }
        function sendEther(address payable receiverAddress) external{
                 // reverts on failure
                // forwards 2,300 gas stipend (not adjustable), safe against reentrancy
                receiverAddress.transfer(1 ether);
        }
}
```

Message to Another Account

address.call.value().gas()()

Account type can be Externally Controlled or Contract

```
pragma solidity ^0.4.0;
contract SimpleContract{
        address payable owner;
        constructor() payable  public{
                owner = msg.sender;
        }
        function sendEther(address payable receiverAddress) external{
                 // returns false on failure
                // forwards all available gas (adjustable), not safe against reentrancy
                bool success;
                bytes memory data;
                 // value and gas are optional
                (success, data) = receiverAddress.call.value(1 ether).gas(3000)("");
        }
}
```

Message to Another Account

Another Contract

```
pragma solidity ^0.4.0;
import RestWhenRich.sol;
contract SimpleContract{
        address payable owner;
        constructor()payable public{
                owner = msg.sender;
        }
        function call(address payable contractAddress) external{
                RestWhenRich  calleeInstance = RestWhenRich(contractAddress);
               // value and gas are optional
                calleeInstance
                        .earnMoney
                        .value(1 ether).gas(50000)("Payer Name");
        }
}
```

More Info: https://solidity.readthedocs.io/en/v0.5.1/control-structures.html#external-function-calls

Fallback Function

- Called when:
  - None of the other functions match the given function identifier
  - Ether sent directly to the contract (without function data)

```
pragma solidity ^0.4.0;
contract SimpleContract{
        address payable owner;
        constructor() public{
                owner = msg.sender;
        }
        function() external payable {
                // Do Something
        }
}
```

More Info: https://solidity.readthedocs.io/en/v0.5.1/contracts.html#fallback-function

Self Destruction

Kill

```
pragma solidity ^0.4.0;
contract SimpleContract{
        address payable owner;
        constructor() public{
                owner = msg.sender;
        }
        function kill() public{
                require(msg.sender == owner);
                // Send contract's balance to owner
                // and delete contract
                selfdestruct(owner);   }

}
```

More Info: https://solidity.readthedocs.io/en/v0.5.1/introduction-to-smart-contracts.html#deactivate-and-self-destruct