Practical session for blockchain

(1) Basics of Solidity - Introduction

Dongkyu Kwak[1] Yeongbong Jin[1]

[1]Financial Risk Engineering Lab.
Seoul National University

April 13, 2022

1. EVM and Solidity
2. Basic Structures - (1) Variables
3. Basic Structures - (2) Contract
4. Some Tiny Examples

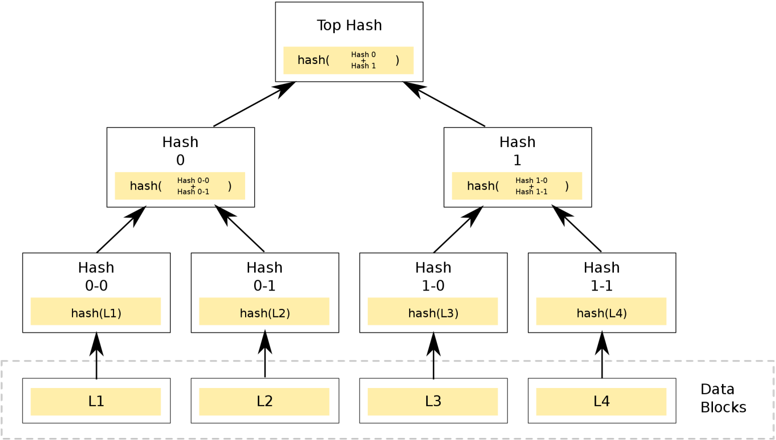
EVM (Ethereum Virtual Machine)

- **EVM(Ethereum Virtual Machine)** is a **computing environment** for users with distributed ledgers in the Ethereum network to distribute and execute **smart contracts**.
- EVM implicitly behaves as a function which maps an old valid **state** and **transactions** to a new valid **state**.

State and Transactions

- **State** is an enormous data structure called a *Merkle Patricia Trie*, which keeps all accounts linked by hashed and reducible to a single root hash stored on the blockchain.
- **Transactions** are cryptographically signed instructions from accounts.

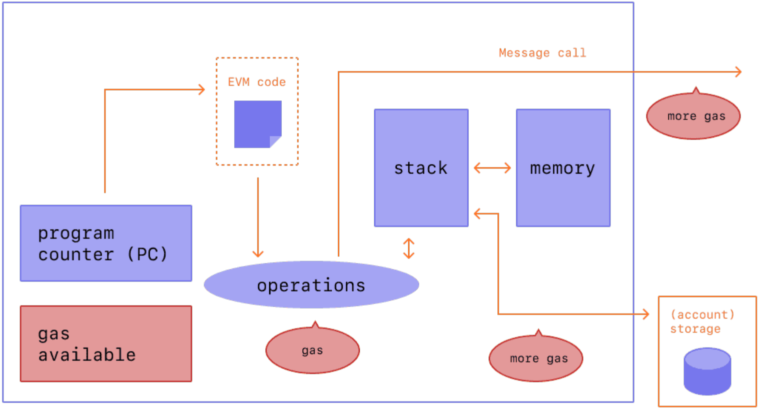
Diagram of Merkle Patricia Trie



EVM (Ethereum Virtual Machine) (cont.)

- Compiled smart contract bytecode excutes as a number of EVM opcodes (XOR, AND, ADD, SUB, etc.) or blockchain-specific stack ops (ADDRESS, BALANCE, BLOCKHASH, etc.)
- Each ops result in **gas** cost, which is the cost for using computing power in the blockchain network
- txs that causes a change in state in **storage** (or contract account) or txs that requires a **message call** incur more gas cost.

Diagram of EVM



Accounts in Ethereum Network

- Ethereum has two account types: **Exteranlly-owned** and **Contract**

Key differences betwen Externally-owned and Contract

- **Externally-owned** account costs nothing at creating, can initiate txs directly, and can only make txs on ETH/token transfers between externally-owned accounts.
- **Contract** account costs gas at creating as it uses network storage, can only send txs in response to receiving a txs.
  Txs from an external account can trigger code executing many different actions, such as transferring tokens or even creating a new contract.

- Both account types have the ability to:
  1. Receive, hold and send ETH and tokens
  2. Interact with deployed smart contracts

Solidity

- **Solidity** is most main-stream high-level language for programming smart contract.
- There are abundant examples on the web for contract, token, and dApp using solidity.

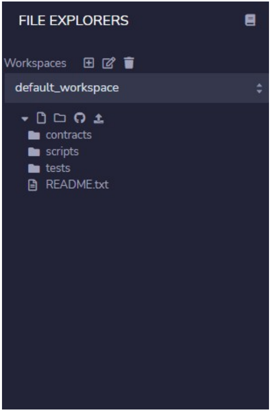
Characteristics of Solidity

- Contract-oriented, supporting Inheritance, Libraries, and other complex user-defined types
- Curly-bracket syntax analogous to C++, JAVA
- Statically typed, that is, the type of a variable is known at compile time

Remix

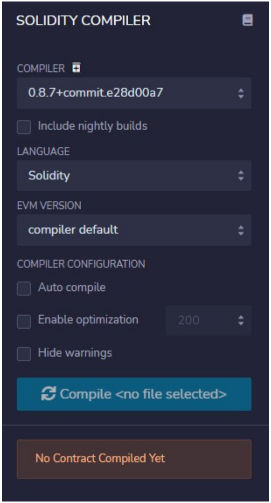
- One needs to use the **Testnet** to test a smart contract before deploying it to Ethereum Mainnet.
- *Remix IDE* provides us with an EVM Testnet environment (JavaScript VM or Web3) for testing smart contracts written via Solidity and Yul.
- Remix IDE helps us to select compiler and EVM version to test smart contracts in their preferred development environment.

Remix - File Exploresr



In *File explorers* section,
you can *upload* a solidity source file (.sol) to the workspace in Remix IDE,
create a new solidity source file,
manage files and folders,
or *download* them.

Remix - Solidity Compiler



SOLIDITY COMPILER

COMPILER

0.8.7+commit.e28d00a7

☐ Include nightly builds

LANGUAGE

Solidity

EVM VERSION

compiler default

COMPILER CONFIGURATION

☐ Auto compile

☐ Enable optimization 200

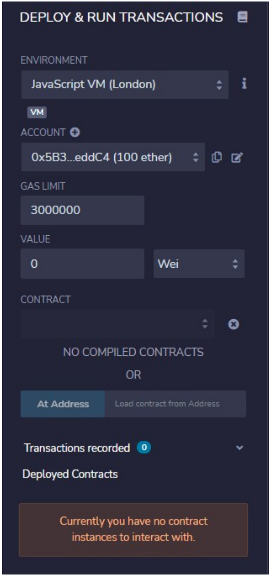
☐ Hide warnings

No Contract Compiled Yet

In *Solidity compiler* section,
you can *compile* a solidity source file (.sol) to a smart contract bytecode.

You can also choose which compiler and EVM version is used.

Remix - Deploy and Run Transactions



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM (London)

VM

ACCOUNT: 0x5B3...eddC4 (100 ether)

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT

NO COMPILED CONTRACTS

OR

At Address | Load contract from Address

Transactions recorded 0

Deployed Contracts

Currently you have no contract instances to interact with.

In *Deploy & run transactions* section, you can *distribute* the compiled smart contract to Testnet or *recall* already deployed contracts.

Furthermore, transactions can be made through a contract, such as *view*, *changing states*, *transferring ETH or token*, *message call* and more.

1. EVM and Solidity
2. Basic Structures - (1) Variables
3. Basic Structures - (2) Contract
4. Some Tiny Examples

Values

- **Booleans**

  bool (true or false)

- **Integer**

  int8, int16, $\cdots$, int256 (n bits signed integers)

  uint8, uint16, $\cdots$, uint256 (n bits unsigned integers)

- **Fixed**

  fixed$M$xN *or* fixed

  (signed float, M : multiple of 8, N integer ranging from 0 to 80;

  Default $(M, N) = (128, 19)$)

  ufixed$M$xN *or* ufixed (unsigned float)

Values (cont.)

- Values can be calculated or compared using operators.

1. **Logical (boolean)**
   ! (NOT), && (AND), || (OR)
2. **Relational operator**
   $<=, \quad <, \quad ==, \quad != , \quad >=, \quad >$
3. **Arithmetic operator**
   $+, \quad -, \quad *, \quad /, \quad **$ (power), % (remainder)

Array

- **Fixed-size array**
  $Type[k]$ (implicit), $[a, b, c, \cdots]$ (literal)

- **Dynamic array**
  $Type[]$

Variable type of elements in an array

An array must contain only variables of the same type.

Array (cont.)

[Ex. 1] Dynamic Array

```
pragma solidity ^0.4.26;

contract testDynamicArray{
    uint[] arr;
    uint public len;

    // Define event to view the values in array through console
    event viewArrayEvt (uint[]);

    // Push an uint value in array
    function pushValue (uint x) public {
        arr.push(x);
        len = arr.length;
    }

    // View the values in array through console
    function viewArray () public {
        emit viewArrayEvt(arr);
    }

}
```

String, Bytes, and Address

- **String, Bytes** (both fixed-size and dynamic)
  (ex. string 'foo', bytes2 0x1ab2, byte[], · · ·)

- **Address**

  40 digits (20 bytes) of hexadecimal bytes

  (ex. 0xdCad3a6d3569DF655070DEd06cb7A1b2Ccd1D3AF)

Variable types of String, Bytes, and Address

String, Bytes, and Address is considered as a type of array, not value.

Structure, Enumerate, Mapping

- **Structure**
  Unlike array, **struct** is a single variable that contains *various types of variables or values* in one fixed structured value.
- **Enumerate**
  enum is a set of predefined constants (max 256 items).
  Each element can be called by uint order.
- **Mapping**
  mapping is an *associative array* that can directly designate the type of a variable that becomes a *key*,
  unlike ordinary array that has only uint type as a key.

Structure, Enumerate, and Mapping (cont.)

[Ex. 2] Enum

```
pragma solidity ^0.4.26;

contract testEnum {
    // Declare and store state enum Switch
    enum MySwitch {
        On,
        Off
    }
    MySwitch public mySwitch;

    // Set default constant state from Enum
    constructor() public {
        mySwitch = MySwitch.Off;
    }

    // Define event to show current state
    event viewSwitchEvt(MySwitch _mySwitch, uint _mySwitchInt);

    function turnOn () public {
        require (mySwitch == MySwitch.Off, "Current switch must be in Off");
        mySwitch = MySwitch.On;
        emit viewSwitchEvt(mySwitch, uint(mySwitch));
    }

    function turnOff () public {
        require (mySwitch == MySwitch.On, "Current switch must be in On");
        mySwitch = MySwitch.Off;
        emit viewSwitchEvt(mySwitch, uint(mySwitch));
    }

}
```

1. EVM and Solidity
2. Basic Structures - (1) Variables
3. Basic Structures - (2) Contract
4. Some Tiny Examples

Contract, State Variables and Functions

- Solidity is a **Contract-Oriented Programming** (COP) language, analogous to Object-Oriented Programming (OOP) languages.
- Thus, every solidity source code must contain **at least one Contract declaration**.
- A contract can be **inherited** from other contracts. (ex. contract A is B, C) That is, all state variables or functions can be duplicated from other contracts without any explicit declaration.

Components of a contract

1. **State variables** : Variables whose values are permanently stored in contract *storage*
2. **Functions** : The executable units of code within a contract

Function Visibility

- Functions have to be specified as being **external**, **internal**, **public**, or **private**.

Components of a contract

1. **external** : External functions can only be called externally.
   (ex. for an external function f, $f()$ does not work, but $this.f()$ works.)
2. **internal** : Internal functions can only be called internally.
   That is, it is only called within current contract.
3. **public** : Public functions are visible from outside of the contract, and can be either called internally or via messages.
4. **private** : Private functions are only visible for the contract where they are defined, and can be executed exclusively within the contract itself.

- Variables can also be formed on **memory** (temporary) or **storage** (blockchain), which costs different gas, respectively.

Constructor, Events and Modifiers

- **Constructor**
  Constructor is a special function that is automatically executed when the contract is deployed.
  It mainly defines *default* values of state variables.
  Constructor must be set *public*.

- **Events**
  Events is used to give an abstraction on top of the EVM' logging functionality.

- **Modifier**
  (Function) **modifier** is an auxiliary element that gives or limits the role of a specific function in advance.

Constructor and Modifiers (cont.)

- There are special modifiers internally featured: **Pure**, **View**, **Return** and **Payable**

Components of a contract

1. **pure** : Pure functions don't access nor change state variables. When pure functions are called externally, it doesn't cost a gas fee.
2. **view** : View functions access state variables but don't change them. When view functions are called externally, it doesn't cost a gas fee.
3. **return** : Return is not a modifier. However, it limits which type of value is returned from the function and lets the compiler know that in advance.
4. **payable** : Payable functions deposit Ether to contract from those called the function. Payable functions are must be externally defined.

1 EVM and Solidity

2 Basic Structures - (1) Variables

3 Basic Structures - (2) Contract

4 Some Tiny Examples

Hello World

[Ex. 3] Hello World

```
pragma solidity ^0.4.26;

contract sayHello{
    string private greeting;

    // Constructor
    constructor () public {
        greeting = "Hello World";
    }

    // Return greet
    function speakHello () public view returns (string){
        return greeting;
    }

    // Change greeting message
    function changeGreet(string _newGreet) public {
        greeting = _newGreet;
    }

}
```

Timer

[Ex. 4] Timer

```
pragma solidity ^0.4.26;

contract Timer{

    uint256 timestamp;
    uint256 setSecond;

    constructor (uint256 _setSecond) public {
        timestamp = block.timestamp * 1 seconds;
        setSecond = _setSecond * 1 seconds;
    }

    // Remained time to expire
    function timeRemained () public view returns (uint256) {
        require(timestamp + setSecond > block.timestamp);
        uint256 callSecond = block.timestamp * 1 seconds;
        uint256 timeRemainedSec = timestamp + setSecond - callSecond;
        return timeRemainedSec;
    }

    function timeOver () public view returns (bool) {
        if (timestamp + setSecond < block.timestamp){
            return true;
        } else {
            return false;
        }

    }

}
```

Contract Account

[Ex. 5] Contract Account

```
pragma solidity ^0.4.26;

contract myContractAccount{
    address public accountOwner;
    uint256 private balance;

    constructor () public {
        accountOwner = msg.sender;
    }
    // Deposit wei(e-18 ether) to contract
    function deposit() public payable {
        require(accountOwner == msg.sender, "Only owner can deposit ether to this account");
        balance += msg.value;
    }
    // Balance check
    function balanceConfirm() public view returns (uint256) {
        require(accountOwner == msg.sender, "Only owner can check the balance of this account.");
        return balance;
    }
    // Withdraw wei(e-18 ether) from contract
    function withdraw(uint _value) public {
        require(accountOwner == msg.sender, "Only owner can withdraw ether from this account");
        require(_value <= balance, "Ether to withdraw must be smaller than the balance.");
        balance -= _value;
        bool sent = accountOwner.send(_value);
        require(sent, "Failed to send Ether");
    }
}
```

Subcurrency

[Ex. 6] Subcurrency

```
pragma solidity ^0.4.26;
contract miniCoin {
    address public minter;
    mapping (address => uint) public balances;
    event Sent(address _from, address _to, uint _amount);
    // Only whose deployed this contract can mint subcurrency
    constructor() public {
        minter = msg.sender;
    }
    // Mint additional amount
    function mint(uint _amount, address _receiver) public {
        require(msg.sender == minter, "Only minter can mint new coin.");
        require(_amount < 1e30);
        balances[_receiver] += _amount;
    }
    // Send minted amount to another holder
    function send(uint _amount, address _receiver) public {
        require(_amount <= balances[msg.sender], "The requested amount to send must be smaller than own ba
        balances[msg.sender] -= _amount;
        balances[_receiver] += _amount;
        emit Sent(msg.sender, _receiver, _amount);
    }
}
```

References

- Basic notions and examples of solidity, https://solidity-kr.readthedocs.io/
- Ethereum developer guide, https://ethereum.org/en/developers/docs/evm/
- Diagram of Merkle Patricia Trie, https://commons.wikimedia.org/wiki/File:Hash_Tree.svg
- Diagram of EVM, https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf