CST4125: Blockchain Development
Week: 15
Title: Introduction to Solidity

Dr Ian Mitchell

smerf.net
Bedfordshire,
UK

Winter2023

Contact and Office Hours

Contact Details

- Name: Dr Ian Mitchell
- Room: TG10
- Address: Middlesex University, Computer Science, London, NW4 4BT
- email: `smerf.net`

Contact and Office Hours

Contact Details

- Name: Dr Ian Mitchell
- Room: TG10
- Address: Middlesex University, Computer Science, London, NW4 4BT
- email: `smerf.net`

Office Hours

- During term time only
- When: Autumn Term: Mondays 1100-1300hrs
- Please read notifications or emails
- There are occassions that these could be arranged online, e.g., due to industrial action or inclement weather

Deadlines

				Feedback	
Description	Submission	Weight	Deadline	Formative	Summative
1. Hyperledger	MyLearning	50%	18th December 2022	LW11-12	12/01/2023
2. Ethereum	MyLearning	50%	2nd April 2023	LW23-24	24/04/2023
Resits	MyLearning	50-100%	1st July 2023	None	None
Deferals	MyLearning	50-100%	1st July 2023	None	None

Lecture Objectives

- Reference v. Value
- Predefined Types
- Data Types
- Licenses
- General Solidity Structure
- **Functions**
  - Storage Rules
  - Events
  - Qualifiers
  - Modifiers
  - Declaration and Scope
  - Parameters
- Events

Ethereum Client

Definition

- Software
- Ethereum Spec.
- P2P
- communicates with other Ethereum Clients
- Interoperate with different clients
- Ethereum is Open Source
- LGPL v3.0
- Define by a formal spec.
- Ethereum Yellow Paper [**wood2014ethereum**]

Ethereum Protocols

- Parity
- Geth
- cpp-ethereum/C++
- pyethereum/python
- censorship resistance?
- Full-node 100Gb
- Local private Network - Ganache
- cloud-base - remix
- Remote-client
- Light-node/client

Full Node

Advantages

- Resilience
- Censorship resistance
- Validates TX
- Disintermediation
- Interact and deploy contracts on public blockchain
- Query blockchain status
- Query blockchain confidentially

Disadvantages

- Significant h/ware
- B/width resources
- Days to sync
- Maintenance and upgrades
- Kept live and online

Public Testnet

Advantages

- less time to sync
- less storage - 10Gb
- test ether, free from faucets
- public blockchains
- running live

Disadvantages

- Not real money
- Cannot test aspects of security
- Not realistic

Local BC Simulation

Advantages

- Ganache
- No sync
- Small amount of data
- You mine first block
- No test ether
- No other users
- No other contracts
- Only the ones you deploy

Disadvantages

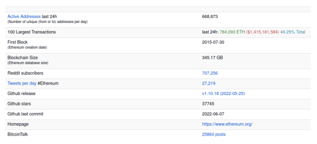
- No other users
- No miners
- No other contracts
- Public contracts

Full Client/Node

System Requirements

Geth

- 100Gb space
- Bandwidth
- SSD-based, I/O intensive
- Ethereum Dimensions
- upto 1Tb of data recommended



Active Addresses last 24h (Number of unique (from or to) addresses per day)	668,873
100 Largest Transactions	last 24h: 784,093 ETH ($1,415,181,584) 44.25% Total
First Block (Ethereum creation date)	2015-07-30
Blockchain Size (Ethereum database size)	345.17 GB
Reddit subscribers	707,256
Tweets per day #Ethereum	27,219
Github release	v1.10.18 (2022-05-25)
Github stars	37745
Github last commit	2022-06-07
Homepage	https://www.ethereum.org/
BitcoinTalk	25860 posts

Full Client/Node

Install

Preliminary

- Git: `sudo apt install git`
- Go: Install instructions
- Geth requires correct version
- Geth: Install instructions
- **or** Parity: Install Instructions

Synchronisation

- Download every block
- Validate every block and transaction
- From genesis block
- Takes time
- DoS attacks in 2016
- Sync goes smoothly upto block 2.28M
- Currently on block 14.93M
- Use `--fast` switch

Remote Clients

- Subset of functionality
- Do not store full blockchain
- Faster and less data
- Manage Private Keys & Addresses
- Create, sign and broadcast TX
- Interact with SCs
- Interact with DApps
- Links to block explorers, e.g. etherscan
- Inject web3

Examples

- Smartphone Wallets
- Browser Wallets

Solidity

Introduction

- Like Javascript & C++
- Statically-typed
- Case-sensitive
- Object-oriented programming (OOP)
- extension `.sol`
- 4 high-level constructs
  1. pragma
  2. comments
  3. import
  4. contracts/library/interface

Pragma

- Directive
- Target compiler version
- Optional
- `pragma solidity 0.6.4`
- Version number: major (6) followed by minor (4) build
- `pragma solidity ^0.4.0`
- Caret is optional:
  - Will use latest version in a major build, so ^0.4.0 would resort to the latest build that is 0.4.19
  - Compile with the major build only 0.4, and not use any other major builds

Comments and imports

Comments

- Single line: `//`
- Multiple line: `/* ... */`
- Natspec: Natural-Specifications-Format

Imports

- `import 'path/filename.sol';`
- Use . for current directory
- Use .. for parent directory
- Use / for seperate directories

Note

Some listings in these slides do not have comments or SPDX identifiers. This is to make the listings clearer and succinct and focus on issues. This is not meant to be repeated. Always leave a blank line before a function. Again these are removed in slides for display purposes.

Licenses

Software Package Data Exchange

- The Linux Foundation
- spdx.org
- Lists all license types
- Easy way to label source code's licenses
- One comment per file
- The first line
- `// SPDX-License-Identifier: MIT`

Why?

- Standardise
- Determine
- Confusion
- Eliminate
- Comments
- List of some Licenses:
  - Apache-2.0
  - EUPL-1.2
  - GPL-3.0
  - MIT

Contract, Libraries & Interface

```
pragma solidity 0.6.19;
// this is a single line comment
/* this is a
multiple line comment */

contract firstContract{

}

contract secondContract{

}

library stringLib{

}

interface IAccount{

}
```

Data Types

Common DataTypes

- `bool`
- `int, int8, int16, ... , int256`
- `uint`
- `fixed, ufixed`
- `address`
- `uint[10]` Byte Array static
- `uint[]` Byte Array dynamic
- `enum`
- `struct`
- `mapping`
- `string`
- `bytes, bytes1, bytes2,`

Literals

- `days`
- `hours`
- `minutes`
- `seconds`
- `wei`
- `szabo`
- `ether`

Predefined Global

Variables

- Message Context
  - `msg.sender`
  - `msg.value`
  - `msg.gas`
  - `msg.data`
  - `msg.sig`
- Transaction context
  - `tx.gasprice`
  - `tx.origin`

- Block Context
  - `block.blockhash`
  - `block.coinbase`
  - `block.difficulty`
  - `block.gaslimit`
  - `block.number`
  - `block.timestamp`

Predefined

Address

- `address.balance`
- `address.transfer(amount)`
- `address.send(amount)`
- `address.call(payload)`
- `address.callcode(payload)`
- `address.delegatecall()`

Functions

- Function declaration

```
function functionName ([parameters])
```

Functions

- Function delimiters and Scope

```
function functionName ([parameters])

 {
 }
```

Functions

- Function access modifiers

```
function functionName ([parameters])
[ public|private|internal|external  ]

 {
 }
```

Functions

- Function return type

```
function functionName ([parameters])
[ public|private|internal|external  ]
[ returns ( data types ) ]
 {
 }
```

Functions

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract ex2{
    uint public age;

    function setAge(uint x) public {
        age = x;
    }

    function getAge() public view returns(uint){
        return age;
    }
}
```

Function Modifiers

- `internal`
- `external`
- default is internal
- Internal function can be called from current contract or inherited contract
- External functions are called by an external account or contract
- Verify before a call to a function, this makes a call to another function before execution of the function

Function Qualifiers

- `constant`: No ability to modify the state of the blockchain. Only read state variables.
- `view`: aliases of constant functions
- `payable`: can accept incoming payments
- `pure`: neither reads or writes any variables in storage.

Pass by Value

- Creates a new memory location
- `x = y;`
- new memory location for both `x` and `y`
- both variables are independent
- change one and the other remains independent
- isolated values

Pass by Reference

- Uses the same memory location
- `x = y;`
- same memory location for both `x` and `y`
- `x` is pointing to same memory location as `y`
- both variables pointing to same memory location
- change value in `x` results in a change in `y`
- change value in `y` results in a change in `x`
- values are not isolated

Variables and storage (adapted from [**modi2018**])

- **Storage**: global memory and permanent storage. Ethereum stores these on every node within its network
- **Memory**: local memory and temporary storage. Will maintain that location for the duration of the function, when function is complete the storage is no longer available.
- **Calldata**: all incoming function execution data is stored. Non-modifiable memory location.
- **Stack**: EVM maintains a stack for loading variables and intermediate values for working with Ethereum instruction set. Stack limit 1024.

Data location

Data storage is dependent on:

- Location of variable declaration
- Data type

The rules?

Storage Rules

Rule 1

Variables declared as state variables are always **Storage**.

Storage Rules

Rule 1

Variables declared as state variables are always **Storage**.

Rule 2

Variables declared as function parameters are always **Memory**.

Storage Rules

Rule 3

Variables declared in functions are by default **Memory**. With some *caveats*:

- value type default is **Memory**.
- reference types default is **Storage**.
- reference types can be overridden
- value types cannot be overridden
- Mappings are by default **Storage**.

Storage Rules

Rule 3

Variables declared in functions are by default **Memory**. With some *caveats*:

- value type default is **Memory**.
- reference types default is **Storage**.
- reference types can be overridden
- value types cannot be overridden
- Mappings are by default **Storage**.

Rule 4

Arguments supplied by callers to function parameters are always stored in **calldata**.

Storage Rules

Rule 5

Assignments to state variables from another state variable are pass by value. They are isolated and independent.

```
pragma solidity >=0.7.0 <0.9.0;
contract ex3{
    uint public x;
    uint public y;
    function setXY(uint a, uint b) public {
        x = a;
        y = b;
    }
    function getModY() public  returns (uint){
        x += y;
        y *= 10;
        return y;
    }
    function getX() public view returns  (uint){
        return x;
    }
```

Rule 5 cont'd

Counter example in Java

```
public class array{
  public int x[] = new int [2];
  public int y[] = new int [2];

  public void setxy(){
    x[0] = 10;
    x[1] = 20;
    y[0] = 1;
    y[1] = 2;
  }
  public void modxy(){
    x = y;
    y[1] = 7;
  }
}
```

```
public class ex5{
public static void main(
    String[] args){
array c = new array();
c.setxy();
c.modxy();
System.out.println("X");
System.out.println(c.x[0]);
System.out.println(c.x[1]);
System.out.println("Y");
System.out.println(c.y[0]);
System.out.println(c.y[1]);
}
}
```

Unfair Comparison?

Are the two comparisons the same?

Storage Rules
Arrays

Rule 5

Assignments to state variables from another state variable are pass by value. They are isolated and independent.

```
contract ex5{
    uint[2] x = [uint(10), 20];
    uint[2] y = [uint(1), 2];
    function getX() public view returns(uint){
        return x[1];
    }
    function getY() public view returns(uint){
        return y[1];
    }
    function ModXY() public returns (uint){
        x = y;
        y[1] = 7;
        return x[1];
    }
```

Storage Rules

Rule 6

Assignments to storage variables from another memory variable always create a new copy.

```
pragma solidity ^0.8.0;
contract ex6{
  uint public stateUint=10;
  function getUint() public returns (uint){
    uint localUint =  20;
    stateUint = localUint;
    localUint = 2;
    return stateUint;
  }
}
```

Rule 6 cont'd
Counter example in Java

```
public class exClass{
  public int x;

  public void setx(){
    x = 57;
  }
}
```

```
public class ex6{
  public static void main(String[] args){
    int localx=100;
    exClass c = new exClass();
    c.setx();//57
    System.out.println(c.x);
    System.out.println(localx);
    c.x = localx;
    localx=3;
    System.out.println(c.x);
    System.out.println(localx);

  }
}
```

Storage Rules

Rule 7

Assignments to memory variables from another state variable always create a new copy.
Converse of Rule 6

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract ex7{
  uint public stateUint = 20;
  event display(uint);
  function create()  public returns (uint) {
    uint localUint = 20;
    localUint = stateUint;
    stateUint=45;
    emit display(localUint);
    return localUint;
  }
}
```

Storage Rules

Rule 8

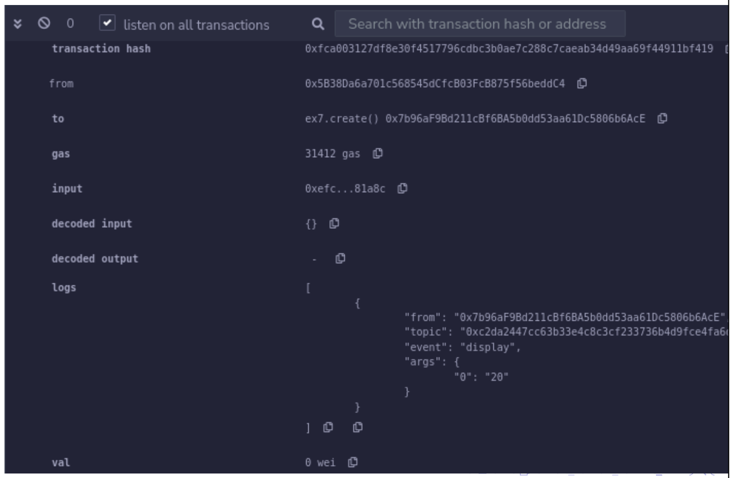
Assignments to memory variables from another memory variable do not create a copy for reference types; however, they do create a new copy for value types.

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.8.0;
contract ex8{
  function test() public pure returns(uint){
    uint a = 25;
    uint b = 31;
    a = b;
    b = 100;
    return a;
  }
}
```

Events

- track execution of a TX sent to a contract
- Dapps can listen to these events
- Events combined with data are recorded as special TX logs on the blockchain
- Can be used as receipts or generally used to display data
- there is no equivalent of Java's `System.out.println`

Events


0
listen on all transactions
Search with transaction hash or address

```
transaction hash    0xfca003127df8e30f4517796cdbc3b0ae7c288c7caeab34d49aa69f44911bf419
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  ex7.create() 0x7b96aF9Bd211cBf6BA5b0dd53aa61Dc5806b6AcE
gas                 31412 gas
input               0xefc...81a8c
decoded input       {}
decoded output       -
logs                [
                        {
                                "from": "0x7b96aF9Bd211cBf6BA5b0dd53aa61Dc5806b6AcE",
                                "topic": "0xc2da2447cc63b33e4c8c3cf233736b4d9fce4fa6
                                "event": "display",
                                "args": {
                                        "0": "20"
                                }
                        }
                    ]
val                 0 wei
```

Reading

- Chapter 3 in [**antonopoulos:2018**]
- Chapter 1 in [**modi2018**]

Summary

- Licenses
- General Solidity Structure
- **Functions**
  - Storage Rules
  - Events
  - Qualifiers
  - Modifiers
  - Declaration and Scope
  - Parameters
- Reference v. Value
- Predefined Types
- Data Types

References I