Solidity [1-3]: Introduction

Solidity [1]: Hello Solidity

What is Solidity?

Solidity is a high-level programming language used to write smart contracts on the Ethereum blockchain

- What are smart contracts?: 사전에 정의된 조건이 만족되면 자동으로 실행되는 프로그램

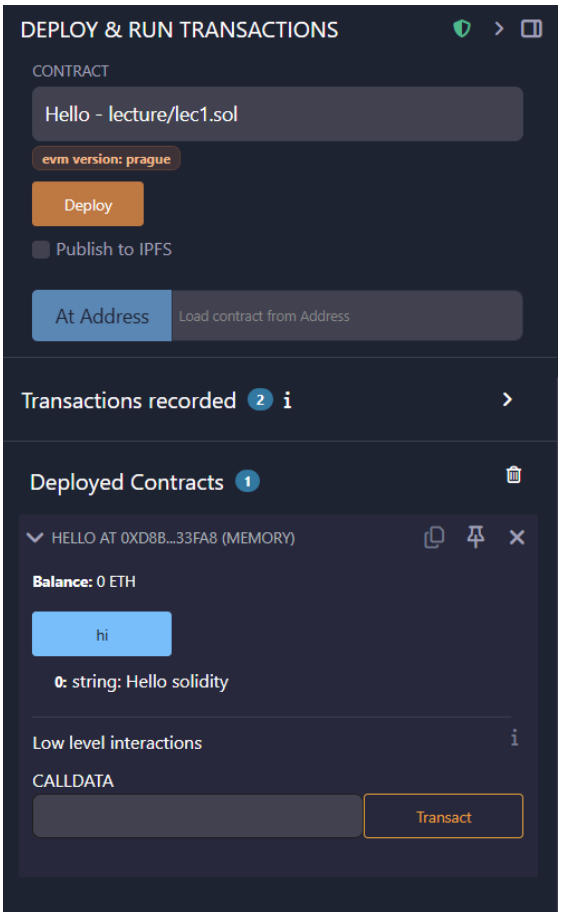
Let’s make simple contract: Hello contract

```
// 첫 줄에 라이선스 명시
// SPDX-License-Identifier: GPL-3.0

// Solidity Compiler 버전 명시
pragma solidity >= 0.7.0 < 0.9.0;

contract Hello {
    string public hi =  "Hello solidity";
}
```

- 첫 줄은 라이선스(저작권 허용 조건)을 명시하는 주석 → 없으면 컴파일러가 경고
- Solidity는 버전이 올라갈수록 **문법 변화**나 **기본 동작 변경**이 자주 있으므로 Solidity 컴파일러 버전 지정



Run 해 준 이후 Deploy 누르면 Deployed Contracts 목록에서 확인 가능

Solidity [2]: Data Type

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.7.0 < 0.9.0;

contract lec2 {
    // Solidity three data type categories:
    // 1. Value types: boolean, bytes, address, unit
    // 2. Reference types :string, Arrays, struct
    // 3. Mapping type: key-value storage

    // ---------- Boolean ----------
    bool public b = false;

    // Logical operations
    bool public b1 = !false;            // true
    bool public b2 = false || true;     // true
    bool public b3 = false == true;     // false
    bool public b4 = false && true;     // false

    // ---------- Bytes ----------
    bytes4 public bt = 0x12345678;
    bytes public bt2 = "STRING";

    // ---------- Address ----------
    address public addr = 0xd8b934580fcE35a11B58C6D73aDeE468a2833f
a8;

    // ---------- Integer ----------
    // int vs uint

    // int8: -128 ~ 127 = (-2^7 ~ 2^7 - 1)
    int8 public it = 4;

    // uint256: = 0 ~ 255 = 0 ~ 2^256 - 1
    uint256 public uit = 132213;

    // uint8: 0 ~ 255 (1바이트, 8비트)
```

```
    uint8 uit2 = 255;
    // uint8 public uit2 = 256;  → Error 발생
}
```

```
(데이터 타입) (접근 제한자) (변수명) = (값)
```

Data types

데이터 타입 종류

- **값 타입(Value types)**: `boolean`, `int`, `uint`, `address`, `bytes` (고정 크기)
- **참조 타입(Reference types)**: `string`, `array`, `struct` (참조로 전달)
- **매핑 타입(Mapping type)**: 키-값 저장 구조

Bytes

- **고정 길이 바이트**: `bytes1` ~ `bytes32`
  - 예: `bytes4 bt = 0x12345678;`
  - 지정한 데이터 타입에서 표현 가능한 수보다 큰 수를 넣으면 자동으로 오류 발생
- **동적 길이 바이트**: `bytes`
  - 예: `bytes bt2 = "STRING";`

Adress

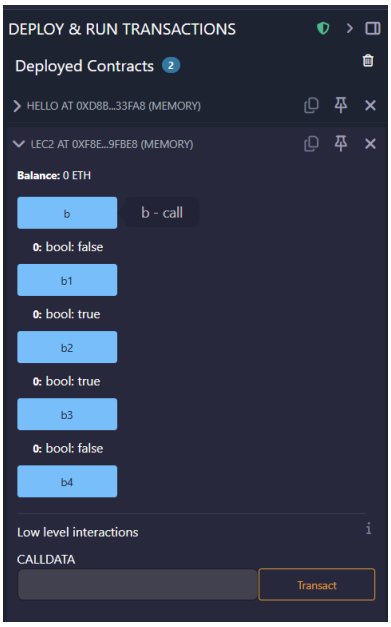
20바이트 이더리움 주소를 저장하는 특수 타입

- 사람 뿐만 아니라 스마트 컨트랙트가 배포 될 때도 adress가 할당됨
- 배포할 때마다 address 주소 달라짐

Integer

- **int**: 부호 있는 정수 (음수/양수 가능)
  - `int8`: -128 ~ 127 (`2^7 ~ 2^7-1`)
  - `uint8`: 0 ~ 255 (`0 ~ 2^8-1`)
- **uint**: 부호 없는 정수 (0 이상만 가능)
  - `uint256`: 0 ~ (2^256 - 1)

Deploy 이후 확인할 수 있는 값들



- 변수마다 들어있는 값과
- 각 deployed transaction에 부여된 address 값도 확인 가능

[3]: ether 단위

이더(Ether) 단위 정리

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.7.0 < 0.9.0;

contract lec3 {
    // 1 ether = 10^9 Gwei = 10 ^ 18 wei
    // 0.000000000000000001 ether = 1 wei
    // 0.01 ether = 10^16wei

    uint256 public value = 1 ether;
    uint256 public value2 = 1 wei;
    uint256 public value3 = 1 gwei;
}
```

Solidity에서 금액을 다룰 때 사용하는 기본 단위는 **wei,** 중간 단위로 **gwei**가 자주 사용됨

기본 단위 관계

- **1 ether = 10^9 gwei = 10^18 wei**

- **1 gwei = 10^9 wei**
- **1 wei = 0.000000000000000001 ether (10^-18 ether)**
- `1 ether` → `1,000,000,000 gwei`
- `1 ether` → `1,000,000,000,000,000,000 wei`
- `0.01 ether` → `10^16 wei`

Gas

Smart Contract를 실행하거나 네트워크와 상호작용할 때 드는 연산 비용

- Gas의 단위는 **Gwei** `1 Gwei = 10^9 wei`
- 특정 연산을 실행하는 데 필요한 Gas의 양은 고정되어 있으며, 이는 **Ethereum Yellow Paper**(p.26)에 정리되어 있음
- 즉, 컨트랙트의 코드가 길어질수록, 또는 더 많은 연산을 수행할수록 더 많은 Gas가 필요함
- Gas 사용 시점: **컨트랙트 배포(Deploy)** 시, 혹은 컨트랙트를 실행하거나 상태를 변경하는 트랜잭션을 보낼 때마다 Gas가 소모됨