What's new in Solidity

ETHonline. October 8, 2021

Toy Problem

- Suppose you want to write a smart contract that sells some object.
- The object has a price.
- There is a limited quantity of the object available.

Problem: A user wants to buy some quantity of the object, and you want to compute the total price.

Solution 1: using regular variables

```
function total_price(uint quantity, uint price) pure returns (uint) {
    return quantity * price;
}
```

Issues

- We want type safety to prevent mixing quantity and price.
- Ideally quantity and price should be two different types.
- The type only represents the underlying data representation and not how the data should be interpreted.

Solution 2: Using structs

```
struct Quantity { uint quantity; }
struct Price { uint price; }

function total_price(
    Quantity memory q,
    Price memory p
) pure returns(uint) {
    return q.quantity * p.price;
}
```

Issues

- Not efficient.
- A struct is a reference type. It is a pointer towards `calldata`, `memory` or `storage`.
- The actual value has to be stored in one of these locations. In the above example: `memory`.

Stack and Memory in EVM

- The EVM is a stack based machine: you do operations using the stack.
  You can push a value to the top and do various operations.
- Memory is a temporary location in EVM where you can store things and read later.
- *Stack is cheaper and more fundamental than memory*.
- First approach: values in stack.
- Second approach: values are in memory.

Stack v/s Memory

- Putting a value in the stack (`push val`): 3 gas.
- Consuming that value: no extra cost.
- Reading a value (copying) from stack: 3 gas.
- Putting a value in memory: `mstore(a, b)`:
  - Putting `b` in stack: 3 gas,
  - Putting `a` in stack: 3 gas,
  - `mstore`: 3 gas (`mstore`) $+ \geq 3$ gas (memory expansion cost),
  - Total: ≥ 12 gas.
- Read a value from memory: `mload(a)`:
  - Putting `a` in stack: 3 gas,
  - `mload`: 3 gas,
  - Total: 6 gas.

User Defined Value Types: a zero cost abstraction

- Can be used from solidity `0.8.9`.
- A way to create an alias.
- Additional type safety.
- Syntax: `type U is V;`
- `U` is the new type.
- `V` is an elementary value type (`uint`, `address`, `int8`, etc.)

Solution 3: User Defined Value Types

```
pragma solidity ^0.8.9;

type Quantity is uint;
type Price is uint;

function total_price(Quantity q, Price p) pure returns(uint) {
    return Quantity.unwrap(q) * Price.unwrap(p);
}
```

- `Quantity.unwrap` for converting `Quantity` to `uint` (the underlying type here.)
- `Quantity.wrap` for converting `uint` to `Quantity`.

Backwards compatibility

```
pragma solidity ^0.8.9;

type Decimal18 is uint256;

interface MinimalERC20 {
    function transfer(address to, Decimal18 value) external;
}

interface AnotherMinimalERC20 {
    function transfer(address to, uint256 value) external;
}
```

Open questions

User defined value types does not have operators right now, but we would like to have them:

```
type Decimal18 is uint256;

// Need a syntax to create operator += for Decimal18

contract MinimalToken {
    mapping (address => Decimal18) public balancesOf;
    function _mint(address user, Decimal18 value) internal {
        // Proof of concept: DOES NOT COMPILE.
        balanceOf[user] += value;
    }
}
```

Participate in the discussion:

1. https://forum.soliditylang.org/t/user-defined-types-and-operators/456
2. https://github.com/ethereum/solidity/issues/11969

Telling a user why a transaction failed

```
contract Vault {
    address immutable owner = msg.sender;
    modifier onlyOwner() {
        // DO NOT DO THIS.
        require(
            owner == msg.sender,
            "The caller was not the owner of the contract."
        );
        _;
    }
    function withdraw() onlyOwner external {
        // do something
    }
}
```

Issues

- Higher deploy cost for contracts.
- Higher runtime cost for reverts.

Custom Errors

```
pragma solidity ^0.8.4;

/// @notice The caller was not the owner of the contract
error OnlyOwner();

contract Ownable {
    address immutable owner = msg.sender;

    modifier onlyOwner() {
        if (msg.sender != owner)
            revert OnlyOwner();
        _;
    }

    function withdraw() onlyOwner external {
        // ...
    }
}
```

Difference

Before

```
modifier onlyOwner() {
    require(
        msg.sender == owner,
        "Ownable: caller is not the owner."
    );
    _;
}
```

After

```
modifier onlyOwner() {
    if (msg.sender != owner)
        revert OnlyOwner();
    _;
}
```

- Cheaper contract deploy cost / smaller bytecode.
- Lower cost for reverting transactions.

Complex revert strings

```
function uint2str(uint i) pure returns (string memory) {
    // ...
}

contract Token {
    mapping (address => uint256) public balanceOf;
    function transfer(address to, uint256 value) external {
        // DO NOT DO THIS!
        require(
            balanceOf[to] >= value,
            string(abi.encodePacked(
                "Insufficient balance for address: ",
                uint2str(uint160(to)),
                ". Current: ",
                uint2str(balanceOf[to]),
                ". Required: ",
                uint2str(value)
            ))
        );
        // ...
    }
}
```

With arguments

```
type Decimal18 is uint256;
/// The user `sender` did not have enough balance.
/// Current balance: `current`.
/// Required balance: `required`.
error InsufficientBalance(
    address sender,
    Decimal18 current,
    Decimal18 required
);
contract Token {
    mapping (address => Decimal18) public balanceOf;
    function transfer(address to, Decimal18 value) external {
        if (Decimal18.unwrap(balanceOf[to]) < Decimal18.unwrap(value))
            revert InsufficientBalance(
                msg.sender,
                balanceOf[to],
                value
            );
          // ...
    }
}
```

Custom Errors: Tooling Support

- Ethers-js
- Hardhat
- Truffle
- Remix
- Dapptools
- . . .

About

Hari.
Solidity team
Slides: `https://hrkrshnn.com/t/ethglobal2021.pdf`
Contact: `https://hrkrshnn.com`
Solidity: `https://soliditylang.org/`