Solidity

Dov Kruger

Department of Electrical and Computer Engineering
Rutgers University

March 27, 2024

Solidity

Solidity is

- A high-level, statically-typed, contract-oriented programming language
- Specifically designed for writing smart contracts.
- Runs on Ethereum Virtual Machine (EVM)
- Influenced by JavaScript, Python, and C++, making it relatively easy for developers to learn and use.
- Enables the creation of decentralized applications (DApps) and protocols.
- Contracts are essentially classes. Solidity is object-oriented.

Installation

- Home page: `https://docs.soliditylang.org/en/latest/installing-solidity.html`
- On Linux: sudo snap install solc

Solidity Development Tools

- **Remix IDE**: An online integrated development environment for Ethereum smart contract development. It allows you to write, compile, deploy, and debug Solidity contracts directly in your web browser.
- **Truffle Suite**: A development framework for Ethereum that provides tools for compiling, testing, and deploying Solidity contracts. It also includes tools for project management and asset handling.
- **Hardhat**: A development environment for Ethereum that offers a wide range of functionalities for building, testing, and deploying smart contracts. It supports tasks such as compiling, testing, and deploying contracts in a scriptable and extensible manner.

The Ethereum Environment

- Ethereum is a blockchain platform focused on smart contracts and decentralized applications (DApps).
- Proposed by Vitalik Buterin in late 2013
- Developed in early 2014, went live on July 30, 2015.
- Ethereum uses blockchain technology, similar to Bitcoin, but with a focus on executing smart contracts.
- The native cryptocurrency of the Ethereum platform is called Ether (ETH), which is used to pay for transaction fees and computational services.
- Ethereum introduces the concept of the Ethereum Virtual Machine (EVM)
- Smart contracts are programs that specify agreements
- Code automatically enforces and executes the terms of the contract but can have bugs of course.
- Ethereum has a thriving developer community and supports multiple programming languages
  - Solidity,
  - Vyper

Comments

- C and C++ comments are supported

```
// This is a comment
/* This is a
multi-line
comment
*/
```

Digital Contracts

- **Immutable**: Once deployed to the blockchain, a digital contract cannot be modified or tampered with. Its code and state are transparent and permanent.
- **Self-executing**: Digital contracts automatically execute predefined conditions and actions when triggered by external inputs or events. This eliminates the need for intermediaries or manual intervention.
- **Trustless**: Digital contracts operate on a decentralized network, such as Ethereum, which removes the need for trust between parties. The contract's execution is verifiable and deterministic, ensuring fairness and transparency.
- **Programmable**: Digital contracts are written in programming languages like Solidity, enabling complex logic and conditions to be encoded into the contract. This allows for a wide range of use cases, from simple transactions to sophisticated financial instruments.
- **Global Reach**: Digital contracts are deployed on a blockchain network accessible worldwide. They can facilitate transactions

Data Types

- Solidity supports various data types including:
  - uint/int - Unsigned and signed integers.
  - address - Ethereum addresses.
  - bool - Boolean values.
  - string - Unicode string data.
  - mapping - Key-value pairs.
  - struct - Custom data structures.
  - array - Fixed and dynamic arrays.

- int
- int8
- int16
- int32
- int64
- int128
- int256

Variables and Constants

- Variables are declared using the `var` keyword.
- Constants are declared using the `constant` keyword.
- Solidity supports state variables, local variables, and function parameters.

Variable Name Rules

- Variable names in Solidity:
    - Must start with a letter (a-z or A-Z) or underscore $(_{)}.Cancontainletters, digits(0-9), andunderscores.$
- Cannot contain spaces or special characters.
- Are case-sensitive.
- Should be descriptive and follow camelCase convention for readability.

Variable Scope

- Solidity allows specifying the visibility of state variables and functions:
  - **public**: Variables and functions can be accessed externally and internally.
  - **internal**: Variables and functions can only be accessed internally (from within the current contract or contracts derived from it).
  - **private**: Variables and functions can only be accessed internally (from within the current contract).

Arithmetic Operators

- Solidity supports various arithmetic operators:
    - + (Addition)
    - − (Subtraction)
    - * (Multiplication)
    - / (Division)
    - % (Modulus)
    - ** (Exponentiation) - Introduced in Solidity 0.6.0

Comparison Operators

- Solidity supports various comparison operators:
  - == (Equal to)
  - != (Not equal to)
  - > (Greater than)
  - < (Less than)
  - >= (Greater than or equal to)
  - <= (Less than or equal to)

Logical (or Relational) Operators

- Solidity supports various logical operators:
  - `&&` (Logical AND)
  - `||` (Logical OR)
  - `!` (Logical NOT)

Assignment Operators

- Solidity supports various assignment operators:
  - = (Simple assignment)
  - += (Addition assignment)
  - −= (Subtraction assignment)
  - ∗= (Multiplication assignment)
  - /= (Division assignment)
  - %= (Modulus assignment)

Conditional (or Ternary) Operators

- Solidity supports the conditional operator:
  - `condition ? value_if_true : value_if_false`
- It is a compact way to write simple conditional statements.

Arrays

Arrays in solidity are similar to C or C++ but with size correctness constraints. Array constants use square brackets

```
uint[] a = [1, 2, 3, 4, 5];
uint[] b = new uint[](5);
for (uint i = 0; i < 5; i++) {
    b[i] = a[i] + 1;
}
uint[] c = [1, 2, 3, 4, 5];
uint[] d = new uint[5];
for (uint i = 0; i < 5; i++) {
    d[i] = c[i] + 1;
}
```

Mappings

Variables and Constants Initialization

```
uint constant myUint = 100;
address constant myAddress = 0x123...456;
bool constant myBool = true;
string constant myString = "Hello, World!";
mapping(uint => string) constant myMapping = {1: "O
uint[] constant myArray = [1, 2, 3, 4, 5];
```

Ether Units

```
assert(1 wei == 1);
assert(1 szabo == 1e12);
assert(1 finney == 1e15);
assert(1 ether == 1e18);
assert(2 ether == 2000 fenny);
```

Hello World Example

```
pragma solidity ^0.8.0;

contract HelloWorld {
  string public greeting;

  constructor() {
    greeting = "Hello, World!";
  }

  function getGreeting() public view returns (string
    return greeting;
  }
}
```

Variables and Constants Initialization

Functions

- Functions are declared using the `function` keyword.
- Solidity supports both public and private functions.
- Functions can return values using the `return` keyword.
- Function modifiers can be used to change the behavior of functions.

Control Structures

- Solidity supports standard control structures:
  - `if`, `else` - Conditional statements.
  - `for`, `while` - Looping constructs.
  - `break`, `continue` - Loop control.

Solidity Programming Constructs

```
// Example of if statement
function exampleIf(uint x) public pure returns (stri
    if (x > 10) {
        return "x is greater than 10";
    } else {
        return "x is not greater than 10";
    }
}

// Example of for loop
function exampleFor(uint n) public pure returns (ui
    uint result = 0;
    for (uint i = 1; i <= n; i++) {
        result += i;
    }
    return result;
}
```