INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Blockchain and Web3 Development

Solidity Types

Aashish Paliwal
International Institute of Information Technology, Hyderabad, India

Table of contents

Value Types

- Boolean
- Integers
- Address
- Fixed sized bytes
- String
- Functions

Reference Types

- Array
- Struct
- Mapping

Types

Solidity is a statically typed language.

Solidity types can be grouped into two categories:

- Value types:
  - Store independent copies of data.
  - When used in:
    - Function arguments → Data is passed by value (copied).
    - Assignments → Data is copied to the new variable.
- Reference types:
  - Store references to the memory location of data.
  - Multiple variables can reference the **same data location**.
  - Used for more complex data types like structs, mapping etc.

Value Types

Value types variables are always passed by value.

Variables of value types do not share memory locations.

Changes to one variable do not affect other variables of the same type.

Currently, value types comprise of: booleans, integers, address, contract types, fixed size byte arrays, strings, enums and function types

Value Types

Boolean

Booleans represented as bool and can have only two values - true and false.

Declared as : bool bool1;

It is possible to explicitly initialize variables during declaration.

Possible operations:

- ! (negation)
- && (and)
- || (or)
- == (equality)
- != (inequality)

Value Types

Integers

Integer number can be represented in two ways:

- uint: unsigned integers. They can only be positive.
- int: signed integers. They can be positive or negative.

Both uint and int are available in increments of 8 bits from 8 up to 256.

Range:

- uint: 0 to 2^n-1
- int: -2^{n-1} to 2^{n-1}-1

The type uint is equivalent to uint256 and the type int is equivalent to int256.

Value Types

Integers

Supported operations:

- Comparisons: <=, <, ==, !=, >=, > (evaluate to bool)
- Bit operators: &, |, ^ (bitwise exclusive or), ~ (bitwise negation)
- Shift operators: << (left shift), >> (right shift)
- Arithmetic operators: +, -, unary - (only for signed integers), *, /, % (modulo), ** (exponentiation)

Value Types

Address

Can be sub-divided into two forms:

- address: can hold a 20 byte value (size of ethereum address).
- address payable: similar to address, but includes additional members, transfer and send.

Key difference: ability to receive ether.

Tip: You can type cast address to address payable: payable(address) [Will come in handy during project].

Operations: <=, <, ==, !=, >= and >.

Most important members of address: balance and transfer

Value Types

Fixed-size byte arrays

The value types bytes1, bytes2, bytes3, …, bytes32 hold a sequence of bytes from one to up to 32.

Declared as bytes32 byteArray = “Just another byte array”;

Operators:

- Comparisons: <=, <, ==, !=, >=, > (evaluate to bool)
- Bit operators: &, |, ^ (bitwise exclusive or), ~ (bitwise negation)
- Shift operators: << (left shift), >> (right shift)
- Index access: If x is of type bytesI, then x[k] for 0 <= k < I returns the k th byte (read-only).

Members: length

Value Types

Strings

Similar to bytes but we don’t have an upper limit on the number of bytes.

Declared as string str = "DemoString";

String is equal to bytes but does not allow length or index access.

Solidity lacks string manipulation functions (even comparison).

How to compare two strings? Hash string and compare.
keccak256(abi.encodePacked(s1)) == keccak256(abi.encodePacked(s2))

Value Types

Functions

There are two kinds of function:

- Internal functions are restricted to being invoked within the current contract.
- External functions are meant to be called by other contracts or transactions.

Declared as

function FuncName (<parameter types>) {internal|external}
[pure|view|payable] [returns (<return types>)]

Reference Types

Reference types contain a reference to the actual data instead of storing the data itself.

Variables of reference types share the underlying data.

Changes made to one variable affect other variables referencing the same data.

Currently, reference types comprise structs, arrays and mappings.

When we use a reference type, we have to explicitly provide the data area where the type is stored: storage, memory and calldata.

Reference Types

Arrays

A group of variable that holds the same data types.

Can be accessed using index.

Array size can be fixed or dynamic.

```
Declared as int[5] arrayName;        // fixed length
Declared as int[] arrayName;         // dynamic length
```

Array members: length, push and pop.

Reference Types

Structs

Custom data types.

Use ‘struct’ keyword to define a structure, consisting of multiple variables (can be a combination of both value and reference type).

Declared as struct Lottery {uint16 lotteryId; address[] participants; bool ended}

[X] aaa.

Reference Types

Mapping

Works similar to hash table.

Mapping is used to store data in the form of key-value pairs.

There is no way to retrieve all keys or values.

There is no way to directly iterate over mapping (without using external storage).

Declared as mapping(address => uint16) public playerPredictions;
Storing data in mapping: playerPredictions[msg.sender] = _predictedNumber;