Introduction to Solidity (Part 1)

Disarankan sudah membaca Algorithm and Data Types (Solidity) untuk pemahaman dasar seputar terminologi dalam pemrograman dan tipe data. Syntax di sini mengikuti versi Solidity terbaru.

Contents

SPDX License Identifier 3
Solidity Version 3
Comments dan NatSpec 3
Inisialisasi Contract 4
Variables 4
State Variable 4
Local Variable 4
Global Variable 4
Operators 5
Arithmetic Operators 5
Logical Operators 5
Comparison Operators 5
Assignment Operators 6
Conditional Operators 6
Function 7
Function Types 7
Function Modifiers 7
Constructor 8
Fallback Functions 8
Data Types 8
Value Types 8
Address 9
Enums 10
Reference Types 10
Array 10
String and Bytes 10
Mapping 11
Struct 11
Units 11
Ether Units 11
Time Units 11
Require Statement 12

SPDX License Identifier

Di Solidity versi 0.6.8, dikenalkan SPDX license identifiers untuk memspesifikasi license yang digunakan contract. Jenis-jenis license identifiers bisa dilihat di https://spdx.org/licenses/. License harus diletakkan di baris pertama contract dan dalam satu contract hanya boleh ada 1 license.

```
// SPDX-License-Identifier: License Name
```

Solidity Version

Solidity memiliki banyak versi, jadi kita perlu memberi tahu compiler versi Solidity yang digunakan dengan keyword **pragma**. Versi Solidity yang didefinisikan hanya berlaku di file contract tersebut, maka pastikan versi yang digunakan saling compatible jika ada lebih dari 1 contract.

```
pragma solidity ^0.8.5; // Di atas versi 0.8.5
pragma solidity >=0.7.0 <0.8.0; // Versi 0.7.0 sampai sebelum 0.8.0.
pragma solidity 0.8.11; // Hanya versi 0.8.11
```

Comments dan NatSpec

Kalian bisa memberikan comment pada contract untuk dokumentasi contract.

```
// Sebaris comment pada contract
/*
Comment
Lebih dari sebaris
*/
```

Solidity juga memiliki comment spesial, yaitu NatSpec (Ethereum Natural Language Specification Format). NatSpec adalah format comment yang memberikan dokumentasi lebih jelas untuk developer dan end-user.

```
/// @title Judul buat contract, library, interface
/// @author Nama penulis
/// @notice Buat ngasih tau kegunaan sesuatu
/// @dev Penjelasan untuk devs
/// @param Penjelasan parameter function atau event
```

Lebih lengkap tentang NatSpec di sini.

NatSpec ini optional, gunakan seperlunya saja. Umumnya, comment biasa sudah cukup.

Inisialisasi Contract

Contract itu hanya sebuah code program yang disimpan di blockchain. File contract dengan bahasa Solidity memiliki extension **.sol**. Untuk penamaan contract, sebaiknya pakai Pascal case dimana huruf pertama setiap kata dikapital, seperti **MintNftContract**, **MyToken**.

```
contract ContractName {

}
```

Variables

Solidity punya 3 jenis variable: state, local, dan global. Berikut yang perlu diperhatikan dalam penamaan variable (hal ini bisa dibilang aturan penamaan secara general):

- Jangan gunakan reserved words seperti **event**, **contract**, **break**.
- Jangan awalai dengan angka (contoh: 12Var), gunakan huruf atau underscore (contoh: _1Var, myString).
- Variable itu case-sensitive, string myString dan string MyString itu 2 variable berbeda.

State Variable

Variable yang nilainya disimpan di contract storage dalam blockchain. State variable punya 3 jenis visibility atau scope.

- Internal, state variable secara **default** bersifat internal. Artinya, variable tersebut hanya bisa diakses oleh **contract yang mendeklarasi dan contract turunannya**.
- Public, artinya variable tersebut bisa diakses **secara internal maupun external** (contract lain atau EOA/external owned account*). Solidity akan by default membuat **getter function**** untuk public state variable.
- Private, state variable hanya bisa diakses oleh contract yang mendeklarasi, **contract turunannya tidak punya akses**.

*EOA itu seperti wallet atau account user.

**Getter function itu function untuk mengambil nilai variable

Local Variable

Variable yang nilainya hanya ada dalam suatu function scope.

Global Variable

Variable spesial yang bersifat global untuk memberikan informasi terkait blockchain dan transaksi. Lebih lengkap tentang global variable di sini.

```
pragma solidity ^0.8.6;
contract MyContract {
   uint counter; // state var
   constructor() {
      counter = 1;
   }
   function addCounter() public returns(address){
      uint a = 1; // local var
      counter += a;
      return msg.sender; // return address of function caller (global var)
   }
}
```

Operators

Di Solidity, juga ada operasi yang dapat dijalankan dalam code dan ini bisa dibilang mirip dengan Bahasa pemrograman lain pada umumnya. Kita bisa bagi ke 6 kategori berikut:

1. Arithmetic Operators
2. Logical Operators
3. Comparison Operators
4. Assignment Operators
5. Bitwise Operators**
6. Conditional Operators

*operan adalah nilai yang dioperasikan

**karena bitwise operators jarang sekali digunakan, maka kalian bisa membaca lebih lengkap di sini.

Arithmetic Operators

Operator	Denotasi	Deskripsi	Syntax
Penjumlahan	+	Menjumlahkan 2 operan	`uint a = b + c;`
Pengurangan	_	Mengurangi operan pertama dengan operan kedua	`uint a = b - c;`
Perkalian	*	Mengalikan 2 operan	`uint a = b * c;`
Pembagian	/	Membagi operan pertama dengan operan kedua	`uint a = b / c;`
Modulus	%	Memberikan sisa dari hasil pembagian operan pertama dengan operan kedua	`uint a = b % 2;`
Increment	++	Menaikkan nilai integer dengan 1	`uint inc = y++;`
Decrement	—	Mengurangi nilai integer dengan 1	`uint dec = x--;`

Logical Operators

Operator	Denotasi	Deskripsi	Syntax
Logical AND	&&	Mengembalikan **true** hanya jika kedua operan **true**, selain itu **false**.	`if (a > 5 && b <= 10)`
Logical OR	||	Mengembalikan **false** hanya jika kedua operan **false**, selain itu **false**.	`if (a > 5 || b <= 10)`
Logical NOT	!	Mengembalikan kebalikannya, jika **true**, maka jadi **false**, dan sebaliknya.	`if (!(a > 5))`

Comparison Operators

Operator	Denotasi	Deskripsi	Syntax
Sama dengan	==	Mengembalikan **true** jika kedua nilai sama, dan sebaliknya.	`bool a = b == c;`
Tidak sama dengan	!=	Mengembalikan **true** jika kedua nilai tidak sama, dan sebaliknya.	`bool a = b != c;`
Lebih besar dari	>	Mengembalikan **true** jika nilai di kiri lebih besar dari yang kanan.	`bool a = b > c;`
Kurang dari	<	Mengembalikan **true** jika nilai di kiri kurang dari yang kanan.	`bool a = b < c;`

Lebih besar dari atau sama dengan	>=	Mengembalikan **true** jika nilai di kiri lebih besar dari atau sama dengan yang kanan.	`bool a = b >= c;`
Kurang dari atau sama dengan	<=	Mengembalikan **true** jika nilai di kiri kurang dari atau sama dengan yang kanan.	`bool a = b <= c;`

Assignment Operators

Seperti Bahasa pemrograman lainnya, variable di Solidity di-assign atau diberi sebuah nilai dengan operator '='. Arithmetic operators bisa dikombinasikan dengan operator '=' untuk menjadi assignment operators. Berikut contoh syntax-nya:

```
uint a = 18; // assign value 18 ke variable a
a += 2; // sama aja dengan a = a + 2; Nilai a jadi 20
a *= 3; // sama aja dengan a = a * 2; Nilai a jadi 60
```

Conditional Operators

Conditional operator ini seperti **if-else** tapi dalam satu baris, sebutan lainnya **ternary operator**. Operator ini memerlukan 3 operan dengan format berikut:

Kondisi **?** ekspresi yang dieksekusi jika **true :** ekspresi yang dieksekusi jika **false**

```
uint a = a > 10 ? a -= 5 : a += 10;
// jika nilai a lebih dari 10, maka akan menjadi a dikurang 5
// tapi jika tidak, maka nilai a akan ditambah 10
```

Function

Function adalah sekelompok code yang dapat dijalankan berkali-kali. Sama seperti state variable, function juga memiliki visibility. Eksekusi function memerlukan gas dalam Ether (native coin).

```
function functionName(parameters) functionTypes returns(returnTypes) {
   // block of code
}
```

Return type bersifat optional. Hanya gunakan jika perlu mengembalikan data.

Function Types

Secara visibility-nya ada 4:

- Public, function bisa dipanggil secara external maupun internal.
- Private, function hanya bisa dipanggil dari contract yang mendeklarasinya.
- External, function hanya bisa dipanggil secara external (contract lain atau EOA). Walau memungkinkan untuk memanggil external function dari dalam contract dengan menambahkan "this" jadi **this.functionName()**, hal ini tidak umum dilakukan.
- Internal, function hanya bisa dipanggil secara internal dari contract yang mendeklarasi dan contract turunan.

Secara behavior-nya ada 3:

- Pure, function tidak mengakses state variable.
- View, function hanya membaca (read) state variable yang menjadi data yang di return.
- Payable, function bisa menerima Ether atau native coin EVM chain.

Function visibility wajib ada, tapi behavior tidak wajib, tergantung functionnya. Kalau tidak memenuhi behavior manapun, tidak perlu ditambahkan.

```
function add(uint256 a, uint256 b) external pure returns(uint256) {
   return a + b;
}
```

Function Modifiers

Function modifiers ini seperti custom function behavior. Kegunaannya untuk membatasi akses, memvalidasi nilai parameter, dan melindungi dari serangan hack. Dalam modifier, ada symbol "_;" yang menandakan function body dari function yang menggunakan modifier. Salah satu modifier yang paling umum digunakan itu **onlyOwner** untuk memastikan yang memanggil function adalah address owner.

```
modifier onlyOwner {
   require(msg.sender == owner);
   _;
}
```

Penjelasan tentang require statement ada di bagian selanjutnya.

Jika function add sebelumnya ingin ditambahkan modifier onlyOwner, maka syntaxnya menjadi:

```
function add(uint256 a, uint256 b) external pure onlyOwner returns(uint256) {
   return a + b;
}
```

Ini artinya function body add menjadi:

```
{
   require(msg.sender == owner);
   return a + b;
}
```

Constructor

Constructor adalah function yang spesial yang akan dijalankan sekali saat contract di-deploy ke blockchain. Penulisan constructor sebelum Solidity v0.4.22 dan v0.4.22 ke atas berbeda. Sejak v0.7.0, visibility untuk constructor juga sudah tidak diperlukan.

```
pragma solidity 0.4.21;
contract MyContract {
   string public message;
   // Constructor
   function MyContract(string _message) public {
      message = _message;
   }
}
```

```
pragma solidity 0.7.0;
contract MyContract {
    string public message;
    // Constructor
    constructor (string _message) {
        message = _message;
    }
}
```

Pada v0.4.22, penulisan bisa menggunakan syntax lama dan baru, tapi sejak v0.4.23, constructor hanya bisa menggunakan syntax baru.

Fallback Functions

Fallback functions juga merupakan function spesial yang visibility-nya harus external dan wajib payable untuk **receive()**. Solidity memiliki 2 jenis fallback functions:

- Fallback, dieksekusi jika function yang dipanggil tidak ada di contract dan tidak ada **receive()**. Optionally payable.
- Receive, dieksekusi jika contract menerima Ether atau native coin EVM chain tanpa ada data (calldata) yang dikirim.

Contract tanpa payable function tidak bisa menerima Ether atau native coin.

```
fallback() external payable {

}
receive() external payable {

}
```

Lebih lengkap bisa lihat modul Akademi Crypto terkait fallback functions.

Data Types

Data types sangat berkaitan dengan ukuran data dan dalam smart contract, storage itu berbayar. Konversi data type disebut typecast.

Keep in mind, 8bits = 1byte.

Value Types

Berikut value types yang dasar.

Type	Keyword	Deskripsi	Syntax
Boolean	bool	Nilainya hanya bisa **true** atau **false**.	`bool isOpen = true;`

Integer	uint8-256 int8-256	Integer ada yang **signed** (bisa angka negative) dan **unsigned** (hanya angka positif dari 0). Biasa kita gunakan **uint**. Angka setelah uint/int adalah ukuran bits-nya. Penulisan uint/int tanpa angka ukuran bits-nya artinya 256bits.*	`uint a = 0;` `int32 b = -10;`
Hexadecimal	address	Ethereum address berukuran 20byte. Bisa contract address atau EOA address. Ada **address** biasa dan **address payable** yang bisa menerima Ether. Address biasa bisa jadi payable dengan typecast **payable(<address>)**.	`address a = msg.sender;` `address payable b;`
Bytes (fixed-size) **	bytes1-32	Untuk menyimpan bytes. Sebelum v0.8.0, **byte** adalah alias untuk **bytes1**.	`bytes4 a = 0x12345678;` `bytes2 a = hex "1234";`

*Ukuran bits-nya menentukan range angka yang dapat disimpan variable dengan data type tersebut.

**Bytes cukup jarang digunakan. Lebih lengkap tentang fixed-size bytes bisa dilihat di sini.

Address

Address berukuran 160bits atau 20bytes jadi uint160 dan bytes20 bisa di-typecast menjadi address dan sebaliknya. Hal ini jarang dilakukan.

```
uint160(msg.sender);
address(uint160(myNum);
```

Data type address memiliki members yang dapat diakses. Semisal ada variable address berikut:

```
address myAddress = 0xdCad3a6d3569DF655070DEd06cb7A1b2Ccd1D3AF;
address payable secAddress = 0x71C7656EC7ab88b098defB751B7401B5f6d8976F;
```

Berikut member dari data type address yang bisa diakses:

```
// balance: cek saldo Ether dalam wei
uint256 balance = myAddress.balance;

// cara megirim Ether ke payable address
// transfer: 2300 gas, kirim error kalau gagal
secondAddress.transfer(20000);

// send: 2300 gas, kirim bool untuk penanda gagal atau sukses
bool isSuccess = secondAddress.send(10000);
require(isSuccess, "Failed to send Ether");

// call: cara yang direkomendasikan untuk mengirim Ether
(bool sent, ) = secondAddress.call{ value: msg.value }("");
require(sent, "Failed to send Ether");
```

Lebih lengkap bisa cek di sini.

Enums

Enum atau enumerable adalah user-defined data type untuk memberi nama ke konstan integer agar contract readability lebih baik. Enum juga membantu membatasi nilai suatu variable. Integer yang diwakili selalu mulai dari 0.

Syntax:

```
enum EnumName {
   element1,
   element2,
   ...,
   elementN
}
```

Contoh:

```
enum Days {
   Monday, // 0
   Tuesday, // 1
   Wednesday, // 2
   Thursday, // 3
   Friday, // 4
   Saturday, // 5
   Sunday // 6
}

Days constant defaultDay = Days.Monday;
```

Reference Types

Reference type memerlukan anotasi data location terkait dimana nilainya disimpan jika dideklarasi sebagai parameter atau di dalam function.

Data location ada 3:

- Storage, disimpan ke blockchain (menjadi state variable).
- Calldata, menyimpan parameter function yang bersifat external dan tidak bisa diubah.
- Memory, menyimpan parameter function selama function dijalankan dan bisa diubah.

Array

Array adalah kumpulan data type yang sama. Array bisa fixed-size atau dynamic-type. Index array mulai dari 0.

```
uint256[] dynamicArr; // jumlah data yang bisa ditampung dinamis
uint256[8] fixedArr; // jumlah data yang bisa ditampung sudah ditentukan
```

Berikut beberapa operasi yang dapat dilakukan pada array:

```
arrayName[index]; // mengakses elemen dalam array
arrayName.length; // mendapatkan panjang array (jumlah elemen)
arrayName.push(value); // menambah elemen array
arrayName.pop(); // membuang elemen array terakhir
```

String and Bytes

String dan bytes adalah array spesial. Data type **bytes** mirip dengan **bytes1[]** dan **string** seperti **bytes** tapi tidak bisa diakses index dan length-nya. Solidity ngga punya function untuk manipulasi string, tapi ada third-party string library.

```
bytes memory a = hex"1234";
a[1]; // 34
a.length; // 2
string myStr = "Abc"; // string adalah kumpulan karakter
```

Mapping

Mapping sifatnya seperti kamus. Gunanya untuk bisa mendapatkan data berdasarkan suatu nilai yang dijadikan kunci (key-value pair).

Syntax:

```
mapping(KeyType KeyName? => ValueType ValueName?) <visibility> <name>;
KeyName dan ValueName bersifat optional dan tidak mempengaruhi fungsionalitas code.
```

Contoh:

```
mapping(address => uint) public balances;
mapping(address user => uint balance) public balances;
balances[msg.sender] += addBalance; // akses dan mengubah value dari key
```

Struct

Struct (structure) adalah user-defined data type dengan banyak property. Struct berguna untuk mengelompokkan data yang saling berhubungan menjadi sebuah data record.

Syntax:

```
struct <StructureName> {
   <data type> var1;
   <data type> var2;
}
```

Contoh:

```
struct Student {
   string name;
   string gender;
   uint studentId;
}
// assign value ke struct instance
Student student = Student("Tengku", "Male", 1);
student.name; // akses property name = Tengku
```

Units

Di Solidity, kita bisa menggunakan unit atau satuan untuk Ether (native coin) dan waktu.

Ether Units

Unit	Deskripsi
wei	Satuan terkecil Ether.
gwei	1 gwei = 1e9 wei.
ether	1 ether = 1e18 wei.

e itu artinya dikali 10^n.

Time Units

Unit	Deskripsi
seconds	Satuan terkecil waktu.
minutes	60 seconds.
hours	60 minutes == 3600 seconds.
days	24 hours == 86400 seconds.
weeks	7 days == 604800 seconds.

Di Solidity v0.5.0, unit **years** dihilangkan karena **setahun tidak tentu 365 days**.

Time units ini bisa digunakan untuk pengecekan waktu dengan **block.timestamp** (unix timestamp).

```
function f(uint start, uint daysAfter) public external {
    if (block.timestamp >= start + daysAfter * 1 days) {
        // ...
    }
}
```

Require Statement

Require adalah statement yang mengecek apakah suatu kondisi terpenuhi sebelum menjalankan code berikutnya dalam suatu function. Balikan dari require statement adalah boolean (true jika kondisi terpenuhi dan sebaliknya). Jika kondisi tidak terpenuhi, maka error message akan dikembalikan jika ada dan sisa gas yang tidak digunakan juga dikembalikan. Error message bersifat optional.

Biasanya require statement digunakan dalam modifier function dan merupakan salah satu cara error handling dalam Solidity yang umum digunakan. Cara error handling lainnya akan dibahas di materi selanjutnya.

Syntax:

```
require(condition, "Error message");
```

Contoh:

```
function transfer(address recipient, uint amount) public {
   require(amount > 0, "Amount must be greater than 0");
   balances[msg.sender] -= amount;
   balances[recipient] += amount;
}
```