In Ethereum Solidity, what is the purpose of the "memory" keyword?

Asked 5 years, 11 months ago Active 25 days ago Viewed 24k times

69

When looking at sample contracts, sometimes arrays are declared in methods with "memory" and sometimes they aren't. What's the difference?

ethereum solidity

26

Share Edit Follow

edited Oct 24 at 0:21

Yilmaz

7,059 6 45 73

asked Nov 21 '15 at 2:48

fivedogit

6,288 5 28 42

3 Answers

Active | Oldest | Votes

94

Without the *memory* keyword, Solidity tries to declare variables in *storage*.

Lead Solidity dev chriseth: "You can think of storage as a large array that has a virtual structure... a structure you cannot change at runtime - it is determined by the state variables in your contract".

That is, the structure of storage is set in stone at the time of contract creation based on your contract-level variable declarations and cannot be changed by future method calls. BUT -- the contents of that storage can be changed with sendTransaction calls. Such calls change "state" which is why contract-level variables are called "state variables". So a variable uint8 storagevar; declared at the contract level can be changed to any valid value of uint8 (0-255) but that "slot" for a value of type uint8 will always be there.

If you declare variables in functions without the *memory* keyword, then solidity will try to use the storage structure, which currently compiles, but can produce unexpected results. *memory* tells solidity to create a chunk of space for the variable at method runtime, guaranteeing its size and structure for future use in that method.

memory cannot be used at the contract level. Only in methods.

See the [the entry "What is the memory keyword? What does it do?"] in the FAQ. I quote it here:

> The Ethereum Virtual Machine has three areas where it can store items.
>
> The first is "storage", where all the contract state variables reside. Every contract has its own storage and it is persistent between function calls and quite expensive to use.
>
> The second is "memory", this is used to hold temporary values. It is erased between (external) function calls and is cheaper to use.
>
> The third one is the stack, which is used to hold small local variables. It is almost free to use, but can only hold a limited amount of values.
>
> For almost all types, you cannot specify where they should be stored, because they are copied everytime they are used.

The types where the so-called storage location is important are structs and arrays. If you e.g. pass such variables in function calls, their data is not copied if it can stay in memory or stay in storage. This means that you can modify their content in the called function and these modifications will still be visible in the caller.

There are defaults for the storage location depending on which type of variable it concerns:

- state variables are always in storage
- function arguments are always in memory
- local variables of struct, array or mapping type reference storage by default
- local variables of value type (i.e. neither array, nor struct nor mapping) are stored in the stack

Share Edit Follow

edited Sep 17 at 6:42

user3755605
103 3

answered Nov 21 '15 at 2:50

fivedogit
6,288 5 28 42

1 Do you have any links to the docs that explain this? I would like to read a bit more on how does the storage works. – Acapulco Dec 8 '15 at 7:33

@Acapuclo It's in the FAQ "What is the memory keyword? What does it do?" – bortzmeyer May 23 '16 at 14:14

2 The FAQ links doesn't work, but if you want to read a similar link I suggest docs.soliditylang.org/en/v0.5.3/... – Quentin Gibson Apr 5 at 17:59

1 I read it but still need a beginner explanation on this, so basically to avoid an expensive operation (save on storage) we should use the `memory` keyword before a function param? If Memory is ephemeral then what's the reason for using it? And how can a contract still call those functions and therefore modify memory once it's already deployed? – Null isTrue Apr 29 at 13:10

1 As someone who hasn't used Solidity it seems bizarre that variables wouldn't be by default in memory and persisting them would be the thing that needs to be explicit – Dominic Jul 28 at 20:57

3

`memory` defines one of the data location in Solidity that can holds the value temporary only during the runtime. the `memory` variables in Solidity can only be declared within the methods and usually used in method parameters. a short term variable that cannot be saved in blockchain... that holds the value only during the execution of a function and its value destroys after execution.

take a look at example function `f()` in which I declared a pointer using `memory` keyword.. it will not alter the value of variable `User`, whereas if it was declared using `storage` it will change the value of variable `User` stored in blockchain and value will not be destroyed...

```
struct User {
 string name;
}
User[] users;

function f() external {
 User memory user = users[0]; // create a pointer
 user.name = "example name" // can't change the value of struct User
}
```

Share Edit Follow

answered Jun 13 at 13:27

Naima Ghulam.M
51 4

1

- Storage holds data between function calls. it is llike computer hard drive. variables are storage data
- Memory temporary place to store data, like RAM. Function args are memory data

Let s say one array is a storage type

```
// state variables are placed in Storage
int[] public numbers

function Numbers()public{
    numbers.push(5)
    numbers.push(10)
    int[] storage myArray=numbers
    // numbers[0] will also be changed to 1
    myArray[0]=1
}
```

`int[] storage myArray=numbers` in this case myArray will point to the same address as "numbers". (it is similar to how referencing objects behave in javascript). See in the function I added 5, then 10 to "numbers" which is placed into Storage. But if you deploy the code on remix and get `numbers[0]` , you will get 1 because of `myArray[0]=1` if you define myArray as memory it will be a different story.

```
// state variables are placed in Storage
int[] public numbers

function Numbers() public{
    numbers.push(5)
    numbers.push(10)
    int[] memory myArray=numbers
    myArray[0]=1
}
```

In this case, "numbers" array is copied into Memory, and myArray now references a memory address which different from "numbers" address. if you deploy this code and reach `numbers[0]` you will get 5.

Share Edit Follow

answered Sep 15 at 21:00

Yilmaz
7,059 6 45 73

Since the `int[] storage myArray` is only a pointer to the numbers variable and no space in storage is reserved for myArray. What's the gas cost for myArray being assigned to numbers ? – Coderboi Oct 12 at 8:16

Also, myArray is a storage reference, so does this pointer is stored in memory or storage itself ? – Coderboi Oct 12 at 8:21
