Solidity

Solidity is a contract-oriented, high-level programming language for implementing smart contracts. Solidity is highly influenced by C++, Python and JavaScript and has been designed to target the Ethereum Virtual Machine (EVM).

Method 1 - npm / Node.js

This is the fastest way to install Solidity compiler on your CentoS Machine. We have following steps to install Solidity Compiler −

Install Node.js

First make sure you have node.js available on your CentOS machine. If it is not available then install it using the following commands −

```
# First install epel-release
$sudo yum install epel-release

# Now install nodejs
$sudo yum install nodejs

# Next install npm (Nodejs Package Manager )
$sudo yum install npm

# Finally verify installation
$npm --version
```

If everything has been installed then you will see an output something like this −

```
3.10.10
```

Install solc

Once you have Node.js package manager installed then you can proceed to install Solidity compiler as below −

```
$sudonpm install -g solc
```

The above command will install solcjs program and will make it available globally through out the system. Now you can test your Solidity compiler by issuing following command −

```
$solcjs-version
```

If everything goes fine, then this will print something as follows −

```
0.5.2+commit.1df8f40c.Emscripten.clang
```

Now you are ready to use solcjs which has fewer features than the standard Solidity compiler but it will give you a good starting point.

Method 2 - Docker Image

You can pull a Docker image and start using it to start with Solidity programming. Following are the simple steps. Following is the command to pull a Solidity Docker Image.

```
$docker pull ethereum/solc:stable
```

Once a docker image is downloaded we can verify it using the following command.

```
$docker run ethereum/solc:stable-version
```

This will print something as follows −

```
$ docker run ethereum/solc:stable -version

solc, the solidity compiler commandlineinterfaceVersion:
0.5.2+commit.1df8f40c.Linux.g++
```

Method 3: Binary Packages Installation

If you are willing to install full fledged compiler on your Linux machine, then please check official website Installing the Solidity Compiler

FIRST APPLICATION

A Solidity source files can contain an any number of contract definitions, import directives and pragma directives.

Let's start with a simple source file of Solidity. Following is an example of a Solidity file −

```
pragma solidity >=0.4.0 <0.6.0;
contract SimpleStorage {
  uint storedData;
  function set(uint x) public {
    storedData = x;
  }
  function get() public view returns (uint) {
    return storedData;
  }}
```

Pragma

The first line is a pragma directive which tells that the source code is written for Solidity version 0.4.0 or anything newer that does not break functionality up to, but not including, version 0.6.0.

A pragma directive is always local to a source file and if you import another file, the pragma from that file will not automatically apply to the importing file.

So a pragma for a file which will not compile earlier than version 0.4.0 and it will also not work on a compiler starting from version 0.5.0 will be written as follows −

```
pragma solidity ^0.4.0;
```

Here the second condition is added by using ^.

Contract

A Solidity contract is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereumblockchain.

The line uintstoredData declares a state variable called storedData of type uint and the functions set and get can be used to modify or retrieve the value of the variable.

Importing Files

Though above example does not have an import statement but Solidity supports import statements that are very similar to those available in JavaScript.

The following statement imports all global symbols from "filename".

```
import "filename";
```

The following example creates a new global symbol symbolName whose members are all the global symbols from "filename".

```
import * as symbolName from "filename";
```

To import a file x from the same directory as the current file, use import "./x" as x;. If you use import "x" as x; instead, a different file could be referenced in a global "include directory".

Reserved Keywords

abstract	after	alias	apply
auto	case	catch	copyof
default	define	final	immutable
implements	in	inline	let
macro	match	mutable	null
of	override	partial	promise
reference	relocatable	sealed	sizeof
static	supports	switch	try
typedef	typeof	unchecked	