**BUILDING DEFI PROTOCOL** [**ERC20 Token Smart Contract for Ethereum Blockchain**](https://www.north-47.com/knowledge-base/erc20-token-smart-contract-for-ethereum-blockchain/)

Since the inception of Blockchain technology; Bitcoin, Ethereum, or crypto-currencies are hot topics and buzzing around the world many startups based on Blockchain technologies are using cryptocurrencies, in other words, crypto tokens for the utilization of their products. These crypto tokens can be deployed on many Blockchain like Ethereum, Cardano, Binance, Polkadot, etc. It’s another topic of discussion, on which blockchain these crypto tokens need to be implemented but as Ethereum being the first market mover, this blog post explains, how you can create such a token on the Ethereum blockchain.

Before creating an Ethereum based token ([ERC20](https://github.com/ethereum/EIPs/issues/20) token), understand first the basics of [Smart-contract](https://ethereum.org/en/developers/docs/smart-contracts/) and their native programming language [Solidity](https://docs.soliditylang.org/en/latest/).

## **Smart Contract**

A smart contract is simply a set of rules that contains the business logic or a protocol according to which all the transactions on a Blockchain should happen. The general purpose of a Smart contract is to satisfy common contractual conditions like creating its token, perform arbitrary competitions, function to send and receive tokens, and store states of transactions.

## **Solidity**

Solidity is an object-oriented and high-level smart-contract programming language, which is developed on top of [Ethereum Virtual Machine](https://ethereum.org/en/developers/docs/evm/) (EVM). Solidity compiler converts smart-contract code into EVM bytecode which is sent to the Ethereum network as a deployment transaction. It would be best to have a good understanding of Solidity programming language to efficiently write an Ethereum Smart Contract and build an application on smart-contract.

## **Coding example of smart-contract**

This section contains the example of a smart-contract code written using the Solidity programming language.

#### **Prerequisite**

##### **Integrated development environment (IDE)**

[Remix](https://remix.ethereum.org/) as the IDE. It is a web-based IDE with built-in static analysis and a testnet EVM. Remix provides the possibility to compile and deploy it to Ethereum testnet with Metamask. [Here](https://hackernoon.com/how-to-use-remix-and-metamask-to-deploy-smart-contracts-on-the-rsk-testnet-zt393xfz) is a good blog post for it.

There is also another web-based IDE available like [EthFiddle](https://ethfiddle.com/" \t "_blank). For more information related to IDE please visit [here](https://ethereum.org/en/developers/docs/ides/).

##### **Programming Language**

Solidity

##### **ERC20 Token Info**

* Symbol – **N47**
* Name – **N47Token**
* Decimals – **0**
* Total Supply – **1000000**

##### **Smart-contract Code**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111 | // SPDX-License-Identifier: unlicensed  pragma solidity 0.8.4;  // ----------------------------------------------------------------------------  // Safe maths  // ----------------------------------------------------------------------------  contract SafeMath {  **function** safeAdd(uint a, uint b) public pure returns (uint c) {          c = a + b;          require(c >= a);      }  **function** safeSub(uint a, uint b) public pure returns (uint c) {          require(b <= a);          c = a - b;      }  }  // ----------------------------------------------------------------------------  // ERC Token Standard #20 Interface  // <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md>  // ----------------------------------------------------------------------------  abstract contract ERC20Interface {  **function** totalSupply() virtual public view returns (uint);  **function** balanceOf(address tokenOwner) virtual public view returns (uint balance);  **function** allowance(address tokenOwner, address spender) virtual public view returns (uint remaining);  **function** transfer(address to, uint tokens) virtual public returns (bool success);  **function** approve(address spender, uint tokens) virtual public returns (bool success);  **function** transferFrom(address from, address to, uint tokens) virtual public returns (bool success);      event Transfer(address indexed from, address indexed to, uint tokens);      event Approval(address indexed tokenOwner, address indexed spender, uint tokens);  }  // ----------------------------------------------------------------------------  // ERC20 Token, with the addition of symbol, name and decimals  // assisted token transfers  // ----------------------------------------------------------------------------  contract N47Token is ERC20Interface, SafeMath {      string public symbol;      string public  name;      uint8 public decimals;      uint public \_totalSupply;      mapping(address => uint) balances;      mapping(address => mapping(address => uint)) allowed;      // ------------------------------------------------------------------------      // Constructor      // ------------------------------------------------------------------------      constructor() {          symbol = "N47";          name = "N47Token";          decimals = 0;          \_totalSupply = 1000000;          balances[msg.sender] = \_totalSupply;          emit Transfer(address(0), msg.sender, \_totalSupply);      }      // ------------------------------------------------------------------------      // Total supply      // ------------------------------------------------------------------------  **function** totalSupply() public override view returns (uint) {  **return** \_totalSupply - balances[address(0)];      }      // ------------------------------------------------------------------------      // Get the token balance for account tokenOwner      // ------------------------------------------------------------------------  **function** balanceOf(address tokenOwner) public override view returns (uint balance) {  **return** balances[tokenOwner];      }      // ------------------------------------------------------------------------      // Transfer the balance from token owner's account to receiver account      // - Owner's account must have sufficient balance to transfer      // - 0 value transfers are allowed      // ------------------------------------------------------------------------  **function** transfer(address receiver, uint tokens) public override returns (bool success) {          balances[msg.sender] = safeSub(balances[msg.sender], tokens);          balances[receiver] = safeAdd(balances[receiver], tokens);          emit Transfer(msg.sender, receiver, tokens);  **return** **true**;      }      // ------------------------------------------------------------------------      // Token owner can approve for spender to transferFrom(...) tokens      // from the token owner's account      //      // <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md>      // recommends that there are no checks for the approval double-spend attack      // as this should be implemented in user interfaces      // ------------------------------------------------------------------------  **function** approve(address spender, uint tokens) public override returns (bool success) {          allowed[msg.sender][spender] = tokens;          emit Approval(msg.sender, spender, tokens);  **return** **true**;      }      // ------------------------------------------------------------------------      // Transfer tokens from sender account to receiver account      //      // The calling account must already have sufficient tokens approve(...)-d      // for spending from sender account and      // - From account must have sufficient balance to transfer      // - Spender must have sufficient allowance to transfer      // - 0 value transfers are allowed      // ------------------------------------------------------------------------  **function** transferFrom(address sender, address receiver, uint tokens) public override returns (bool success) {          balances[sender] = safeSub(balances[sender], tokens);          allowed[sender][msg.sender] = safeSub(allowed[sender][msg.sender], tokens);          balances[receiver] = safeAdd(balances[receiver], tokens);          emit Transfer(sender, receiver, tokens);  **return** **true**;      }      // ------------------------------------------------------------------------      // Returns the amount of tokens approved by the owner that can be      // transferred to the spender's account      // ------------------------------------------------------------------------  **function** allowance(address tokenOwner, address spender) public override view returns (uint remaining) {  **return** allowed[tokenOwner][spender];      }  } |

Using the above code, smart-contract can be deployed on Ethereum [Mainnet or Testnet](https://ethereum.org/en/developers/docs/networks/). Deploying a smart contract is technically a transaction, that needs to pay [Gas](https://ethereum.org/en/developers/docs/gas/) (fees) in terms of ETH (Native token for Ethereum network), in the same way, that needs to pay gas for a simple ETH transfer. However, Gas costs for contract deployment are far higher.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated









