**Activity – ERC 20 Tokens**

**Part 1.**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol";

contract ERC20 is IERC20 {

mapping (address => uint256) private \_balances;

mapping (address => mapping (address => uint256)) private \_allowed;

uint256 private \_totalSupply;

string private \_name;

string private \_symbol;

function totalSupply() public view override returns (uint256) {

return \_totalSupply;

}

function balanceOf(address owner) public view override returns (uint256) {

return \_balances[owner];

}

function allowance(address owner, address spender) public view override returns (uint256){

return \_allowed[owner][spender];

}

function transfer(address to, uint256 value) public override returns (bool) {

require(value <= \_balances[msg.sender]);

require(to != address(0));

\_balances[msg.sender] = \_balances[msg.sender] - value;

\_balances[to] = \_balances[to] + value;

emit Transfer(msg.sender, to, value);

return true;

}

function approve(address spender, uint256 value) override public returns (bool) {

require(spender != address(0));

\_allowed[msg.sender][spender] = value;

emit Approval(msg.sender, spender, value);

return true;

}

function transferFrom(address from, address to, uint256 value ) public override returns (bool)

{

require(value <= \_balances[from]);

require(value <= \_allowed[from][msg.sender]);

require(to != address(0));

\_balances[from] = \_balances[from] - value;

\_balances[to] = \_balances[to] + value;

\_allowed[from][msg.sender] = \_allowed[from][msg.sender] - value;

emit Transfer(from, to, value);

return true;

}

function \_mint(address account, uint256 amount) internal {

require(account != address(0));

\_totalSupply = \_totalSupply + amount;

\_balances[account] = \_balances[account] + amount;

emit Transfer(address(0), account, amount);

}

}

The *ERC20* contract as given above has no means to create token supply.

1. Add a constructor to initialize it with a name, symbol and initial supply.
2. However even the this constructor, currently there is no easy way to transfer tokens out to any EOA. Modify the constructor so that the contract owner account gets all the initial token supply.

Test that your constructor is working correctly. You can use the Javascript VM environment.

Ok now we are set. Test out the following :

1. *Transfer* some token to another account using the transfer() function. Use the balance() function to check that the amount has been sent.

Fill in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Transfer() | Account \_from | Account \_to | Msg.sender |
| Account number |  |  |  |
| Balance |  |  |  |

1. Test out the *transferFrom*() function. How would you test it?

Fill in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Approve()** | Token Owner Account | Delegate Account (spender) | Msg.sender |
| **Account Number** |  |  |  |
| **Balance** |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Allowance()** | Token Owner Account | Delegate Account (spender) | Msg.sender |
| **Account Number** |  |  |  |
| **Allowance Value** |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **transferFrom ()** | Account From | Account To | Msg.sender |
| **Account Number** |  |  |  |
| **Balance** |  |  |  |

After transferFrom() call

|  |  |  |  |
| --- | --- | --- | --- |
| **Allowance()** | Token Owner Account | Delegate Account (spender) | Msg.sender |
| **Account Number** |  |  |  |
| **Allowance Value** |  | | |

1. Create an external function mint() that will add to the token supply. Note: creating extra tokens does not require mining at all.

**Part 2**

For this activity you will work in a group. There will be one contract owner and others as other account holders.

Now it is time to test it on the Ethereum testnet (you can use Kovan or Ropsten). Make sure you have some Eth – get them through the testnet faucet. Google for them.

Set remix to use Injected Web3 after you have login to Metamask. Create some accounts (3) with some Eth balance.

Deploy your amended ERC20 contract on the testnet. Wait for the confirmations. Copy the deployment address.

We want to test sending tokens to other accounts on other metamask accounts (on different computers). Find a partner and get her account address. At the same time give her your contract deployment address.

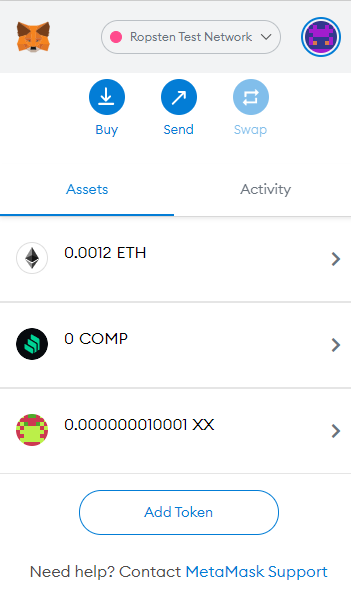


Contract Owner

Contract Address

Account Address

Transfer some tokens to your friend’s account. Check that the balance have been updated on the account that you have transferred token to. Your friend should be able to see her tokens (scroll down till you see the Add Token link) after adding the token. She will need the token contract address.



Your friend should also be able to access the same token contract by starting remix and loading the contract address.

Before that you need to send the contract abi (application binary interface) file to the other account. The ABI specifies the interface of the contracts, i.e. what functions are available.



Contract Owner

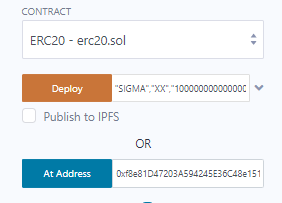
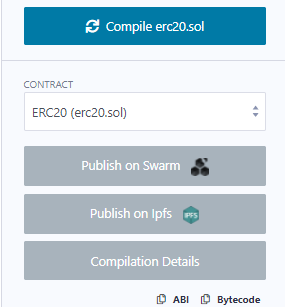
erc20.abi

Create a new file erc20.abi

Load contract address

Here is what you need to do:

1. Copy the abi at the Solidity compiler page. See picture below.
2. Save it in a file named something.abi (eg erc20.abi)
3. Send this file to the other user (ie your friend).
4. Your friend now creates a new file in Remix with extension \*.abi and copy the ABI content to it.
5. Make sure this file is the active tab in the editor. At Deploy & Run page, in the field next to At Address, input the contract’s address and click on At Address. If successful, an instance of the contract will appear below - in the list of Deployed Contracts



Now your friend can access the same contract across the blockchain.

Please run through the transferFrom function again. Check that you understand the concept again.

|  |  |  |  |
| --- | --- | --- | --- |
| **Approve()** | Token Owner Account | Delegate Account (spender) | Msg.sender |
| **Account Number** |  |  |  |
| **Balance** |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Allowance()** | Token Owner Account | Delegate Account (spender) | Msg.sender |
| **Account Number** |  |  |  |
| **Allowance Value** |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **transferFrom ()** | Account From | Account To | Msg.sender |
| **Account Number** |  |  |  |
| **Balance** |  |  |  |

After transferFrom() call

|  |  |  |  |
| --- | --- | --- | --- |
| **Allowance()** | Token Owner Account | Delegate Account (spender) | Msg.sender |
| **Account Number** |  |  |  |
| **Allowance Value** |  | | |

**Part 3**

**Creating a token decentralized exchange DEX**

For this activity, we’ll create a simple decentralized exchange DEX where a user can trade Ethereum with our newly deployed ERC-20 token. Our DEX will use an instance of the ERC20 token contract in its constructor. The DEX contract will have functions for exchanging tokens to ether and exchanging ether to tokens. The contract has two functions:

* buy: The user can send ether and get tokens in exchange
* sell: The user can decide to send tokens to get ether back

To keep things simple, we just exchange 1 token for 1 ether.

The basic structure of the contract:

import "erc20.sol";

contract DEX {

ERC20 public token;

event Bought(uint256 amount);

event Sold(uint256 amount);

//actual contract address

address erc20add = 0x9a2E12340354d2532b4247da3704D2A5d73Bd189;

constructor() {

token = ERC20(erc20add); //load existing contract, not create new

}

function buy() payable public { //buy tokens with Eth

// TODO

}

function sell(uint256 amount) public { //sell tokens in return with Eth

// TODO

}

}

The DEX contract should have some tokens to start of with. The **buy** function will transfer Eth from the buyer to the DEX contract and receive tokens from DEX contract. The sell function will receive Eth from the DEX contract and the DEX will receive tokens.

**The Buy Function**

For the buy function we need to first need to check the amount of tokens that the contracts own is sufficient. The amount of Eth used is available at the msg.value of the transaction. If the contract owns enough tokens it’ll send the number of tokens to the user and emit the Bought event.

**The Sell Function**

The function responsible for the sell will first require the user to have approved the amount of token to transfer by calling the erc20 approve function beforehand. Then when the sell function is called, the contract check if the approval from the token owner is correct. Then the token contract will transfer the token from the seller to the DEX contract.

Your task is to code both those functions.



Buy: Receive Token, Give ETH

Sell: Give Token, Receive ETH

DEX Contract

Token

Contract