Smart Contracts Exercise 03: ERC-20 CTU Token

1 Introduction

Tokens in the Ethereum ecosystem are smart contracts that implement a standardized interface. They are designed to virtually represent certain assets. For example, financial assets such as shares in a company, use as cryptocurrencies (stable coins like USDC, DAI), allow holders to vote on decisions in decentralized projects (e.g., Uniswap - UNI), enable artists to tokenize their works and sell them as unique digital items (NFTs), represent collectibles in games, or are used for digital identity or access to services. Depending on the use case, there are different types of tokens, each serving different purposes. Below are three common types of tokens:

1. **Fungible Tokens**

   Fungible tokens (*ERC-20 Tokens*) are interchangeable and have exactly the same value. Each unit of a fungible token is identical to another unit. Examples are cryptocurrencies, utility tokens or governance tokens.

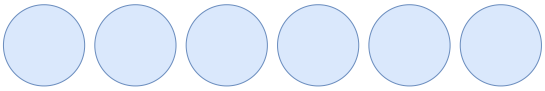

Figure 1: Fungible Tokens

2. **Non-Fungible Tokens (NFTs)**

   Non-fungible tokens (*ERC-721 Tokens*) are unique and cannot be exchanged on a one-to-one basis. They are used to represent ownership of unique items such as digital art, collectibles, and real estate. Each token is uniquely identifiable by an ID.

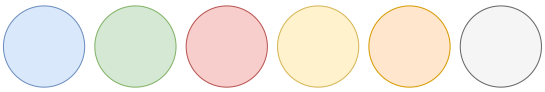

Figure 2: Non-Fungible Tokens

3. **Multi-Tokens**

   Multi-tokens (*ERC-1155 Tokens*) combine the properties of both fungible and non-fungible tokens. They allow for the creation of multiple token types within a single contract, providing flexibility for various use cases.

Figure 3: Multi-Tokens

In this exercise, you will create your own ERC-20 token contract according to the specified standard, and then we will attempt to hack this contract together.

Prerequisites

Ensure that you have already installed the following on your system:

- **Node.js** - https://nodejs.org/en/ An open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.
- **NPM**: Node Package Manager, which comes with Node.js.

Open your terminal and run the following commands to verify the installations:

```
$ node -v
$ npm -v
```

Both commands should return the installed version numbers of Node.js and NPM respectively. Node.js provides the runtime environment required to execute JavaScript-based tools like Hardhat, while NPM is used to manage the packages and dependencies needed for development. It is recommended that you use NPM 7 or higher.

Project Set Up

To get started, visit the following GitLab repository and clone it to your local machine. This repository contains a template in which you will complete this exercise. After you clone the repository start with the following command within your project folder:

```
$ npm install
```

This will install all the necessary dependencies for the project. Your implementation will be in the file *contracts/CTUToken.sol*. In this file, there are #TODO comments where you should implement the required functionality. To fulfill this task you need to pass all the provided tests. You can run the tests with the following command:

```
$ npx hardhat test
```

2 Specification: ERC-20 Token

The ERC-20 standard was first proposed by Fabian Vogelsteller and Vitalik Buterin in November 2015. The token specification defines the interface that a smart contract must implement to be ERC-20 compliant. It is important to note that it **does not specify**

the actual implementation. It is the most widely used standard with more than 1.5 million smart contracts on the main net implementing it.

Example functionalities ERC-20 provides:

- Transfer tokens from one account to another.
- Get the current token balance of an account.
- Get the total supply of the token available on the network.
- Approve whether an amount of token from an account can be spent by a third-party account.

If a Smart Contract implements the following methods and events, it can be called an ERC-20 Token Contract. Once deployed, it will be responsible for keeping track of the created tokens on Ethereum. To see the full specification, visit ERC-20 Token Standard.

Methods

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value)
  public returns (bool success)
function approve(address _spender, uint256 _value) public returns (bool success)
function allowance(address _owner, address _spender)
  public view returns (uint256 remaining)
```

Events

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

OpenZeppelin

OpenZeppelin provides a library for secure smart contract development. It is built on a solid foundation of community-vetted code. It is good practice to use standardized implementations like those from OpenZeppelin. Documentation about available contracts made by Open Zeppelin can be found here. The actual implementations of the contracts are available on GitHub. OpenZeppelin contracts can be installed using npm and imported directly into a contract. The ERC20 implementation by OpenZeppelin is a standard recognized by the official EIP20 documentation. You can find the implementation here. It is a common practice to use the ERC20 implementation by OpenZeppelin when creating ERC20 token contracts, instead of explicitly implementing the ERC20 interface inside the contract. However, for the educational purpose of this exercise, you will implement the ERC20 contract by yourself. The implementation of your the CTU Token contract using OpenZeppelin would look like this:

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// Import OpenZeppelin's ERC20 implementation
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

/**
 * @title CTUToken
 * @dev A custom implementation of an ERC-20 Token using OpenZeppelin's library.
 */
contract CTUToken is ERC20 {
    // Define the initial supply: 1,000,000 tokens with 18 decimal places
    uint256 private constant INITIAL_SUPPLY = 1_000_000 * 10 ** 18;

    /**
     * @dev Constructor that initializes the ERC-20 token with a name and symbol,
     * and mints the total supply to the deployer's address.
     */
    constructor() ERC20("CTU Token", "CTU") {
        // Mint the initial supply to the deployer of the contract
        _mint(msg.sender, INITIAL_SUPPLY);
    }
}
```

3 CTU Token