### **LibertyCoin Smart Contract**

solidity

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import "@openzeppelin/contracts/access/Ownable.sol";

contract LibertyCoin is ERC20, Ownable {

// Initial supply of tokens

uint256 private constant INITIAL\_SUPPLY = 1\_000\_000\_000 \* 10 \*\* 18; // 1 billion tokens with 18 decimals

uint256 public exchangeRate = 100; // Example: 1 LIBERTY = 0.01 ETH (100 LIBERTY = 1 ETH)

constructor() ERC20("LibertyCoin", "LIBERTY") {

\_mint(msg.sender, INITIAL\_SUPPLY);

}

// Function to mint new tokens (only owner can call this)

function mint(address to, uint256 amount) public onlyOwner {

\_mint(to, amount);

}

// Function to burn tokens

function burn(uint256 amount) public {

\_burn(msg.sender, amount);

}

// Function to exchange LibertyCoin for ETH

function exchange(uint256 amount) public {

require(amount > 0, "Amount must be greater than zero");

require(balanceOf(msg.sender) >= amount, "Insufficient balance");

uint256 ethAmount = amount / exchangeRate;

require(address(this).balance >= ethAmount, "Insufficient contract balance");

\_burn(msg.sender, amount);

payable(msg.sender).transfer(ethAmount);

}

// Function to receive ETH

receive() external payable {}

}

### **Changes Explained**

* Initial Supply: The INITIAL\_SUPPLY constant is set to 1 billion tokens (1\_000\_000\_000).
* Exchange Rate: A public variable exchangeRate is added to define the rate for exchanging LibertyCoins for ETH. In this example, 100 LIBERTY equals 1 ETH.
* Exchange Function:
  + The exchange function allows users to exchange their LibertyCoins for ETH.
  + It checks that the user has enough tokens and that the contract has sufficient ETH to fulfill the exchange.
  + The tokens are burned upon exchange, and the equivalent ETH is sent to the user's address.
* Receive Function: A receive function is added to allow the contract to accept ETH.

### 

### **Important Considerations**

* Security: This is a simplified example. In a production contract, consider implementing security measures such as rate limiting, and checks to prevent reentrancy attacks.
* Liquidity: For the exchange functionality to work, the contract must have ETH in its balance. Ensure you fund the contract appropriately.
* Testing: Thoroughly test the contract on a testnet before deploying it on the mainnet.