// SPDX-License-Identifier: UNLISCENSED

pragma solidity 0.8.4;

/\*\*

\* @title SampleBEP20Token

\* @dev Very simple BEP20 Token example, where all tokens are pre-assigned to the creator.

\* Note they can later distribute these tokens as they wish using `transfer` and other

\* `BEP20` functions.

\* USE IT ONLY FOR LEARNING PURPOSES. SHOULD BE MODIFIED FOR PRODUCTION

\*/

contract SampleBEP20Token {

string public name = " ABLE COIN";

string public symbol = "ABC";

uint256 public totalSupply = 50000000000000000000; //

uint8 public decimals = 8;

/\*\*

\* @dev Emitted when `value` tokens are moved from one account (`from`) to

\* another (`to`).

\*

\* Note that `value` may be zero.

\*/

event Transfer(address indexed \_from, address indexed \_to, uint256 \_value);

/\*\*

\* @dev Emitted when the allowance of a `spender` for an `owner` is set by

\* a call to {approve}. `value` is the new allowance.

\*/

event Approval(

address indexed \_owner,

address indexed \_spender,

uint256 \_value

);

mapping(address => uint256) public balanceOf;

mapping(address => mapping(address => uint256)) public allowance;

/\*\*

\* @dev Constructor that gives msg.sender all of existing tokens.

\*/

constructor() {

balanceOf[msg.sender] = totalSupply;

}

/\*\*

\* @dev Moves `amount` tokens from the caller's account to `recipient`.

\*

\* Returns a boolean value indicating whether the operation succeeded.

\*

\* Emits a {Transfer} event.

\*/

function transfer(address \_to, uint256 \_value)

public

returns (bool success)

{

require(balanceOf[msg.sender] >= \_value);

balanceOf[msg.sender] -= \_value;

balanceOf[\_to] += \_value;

emit Transfer(msg.sender, \_to, \_value);

return true;

}

/\*\*

\* @dev Sets `amount` as the allowance of `spender` over the caller's tokens.

\*

\* Returns a boolean value indicating whether the operation succeeded.

\*

\* IMPORTANT: Beware that changing an allowance with this method brings the risk

\* that someone may use both the old and the new allowance by unfortunate

\* transaction ordering. One possible solution to mitigate this race

\* condition is to first reduce the spender's allowance to 0 and set the

\* desired value afterwards:

\* https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

\*

\* Emits an {Approval} event.

\*/

function approve(address \_spender, uint256 \_value)

public

returns (bool success)

{

allowance[msg.sender][\_spender] = \_value;

emit Approval(msg.sender, \_spender, \_value);

return true;

}

/\*\*

\* @dev Moves `amount` tokens from `sender` to `recipient` using the

\* allowance mechanism. `amount` is then deducted from the caller's

\* allowance.

\*

\* Returns a boolean value indicating whether the operation succeeded.

\*

\* Emits a {Transfer} event.

\*/

function transferFrom(

address \_from,

address \_to,

uint256 \_value

) public returns (bool success) {

require(\_value <= balanceOf[\_from]);

require(\_value <= allowance[\_from][msg.sender]);

balanceOf[\_from] -= \_value;

balanceOf[\_to] += \_value;

allowance[\_from][msg.sender] -= \_value;

emit Transfer(\_from, \_to, \_value);

return true;

}

}