成都链安
BEOSIN

Smart contract security audit report

Audit Number：202001101812

Source code： https://github.com/FinNexus/FinNexus-token

commit hash： d2d054ce9acfe64a5c9f645cd25a7d755bd11ebc

Smart Contract Name：

index	File Name
1	contracts\CfncToken.sol
2	contracts\ERC20Protocol.sol
3	contracts\FinNexusContribution.sol
4	contracts\Owned.sol
5	contracts\SafeMath.sol
6	contracts\StandardToken.sol
7	contracts\UM1SToken.sol

Start Date：2020.01.07

Completion Date：2020.01.10

Overall Result：Pass（Merit）

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass

		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		selfdestruct Function Security	Pass
3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	**Missing**
12	Fake Deposit	-	**Fail**
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of project FinNexus-token, including Coding Standards, Security, and Business Logic. **FinNexus-token did not pass all audit items. It failed at Fake Deposit, which does not affect the normal token transfer. The overall result is Pass (Merit).** Please find below the basic information of the smart contract:

1、Basic Token Information

Token name	CfncCoin (Changeable)
Token symbol	CFNC (Changeable)
decimals	18
totalSupply	0 (Mintable, up to 500 million)
Token type	ERC20

Table 1 – Basic Token Information of CFNC

Token name	UM1SCoin
Token symbol	UM1S
decimals	18
totalSupply	0 (Mintable, up to 400 million)
Token type	ERC20

Table 2 – Basic Token Information of UM1S

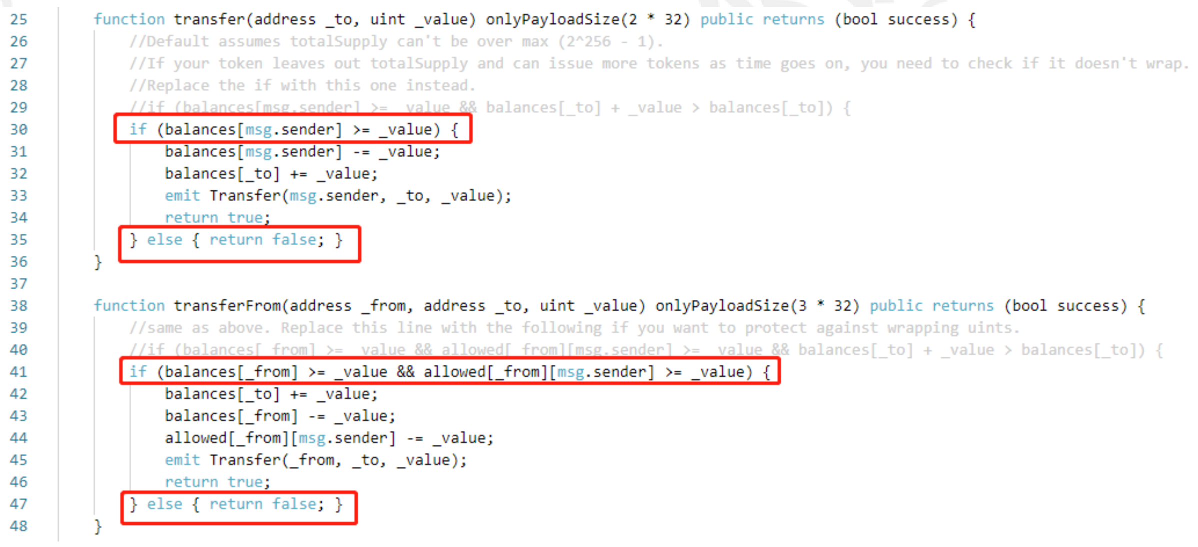
2、Token Vesting Information

Missing

Detailed explanations of the 'Fails' in Results:

➢ Fake Deposit

In the Wan chain token transaction receipt, the status field is 0x01 (true) or 0x00 (false). When the user invokes the function of contract, if the function does not throw an exception, the status of the transaction is 0x01 (true), otherwise, the status is 0x00 (false). As shown in Figure 1 below, the transfer and transferFrom functions in the StandardToken contract return false when the condition is not satisfied, and no exception is thrown. As a result, the status field is still true in the failed transfer transaction. If the service platform such as exchange only by judging that 'TxReceipt Status' is success (status field is 0x01) as successful deposit, then there may be a fake deposit vulnerability.



```
function transfer(address _to, uint _value) onlyPayloadSize(2 * 32) public returns (bool success) {
    //Default assumes totalSupply can't be over max (2^256 - 1).
    //If your token leaves out totalSupply and can issue more tokens as time goes on, you need to check if it doesn't wrap.
    //Replace the if with this one instead.
    //if (balances[msg.sender] >= _value && balances[_to] + _value > balances[_to]) {
    if (balances[msg.sender] >= _value) {
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        emit Transfer(msg.sender, _to, _value);
        return true;
    } else { return false; }
}

function transferFrom(address _from, address _to, uint _value) onlyPayloadSize(3 * 32) public returns (bool success) {
    //same as above. Replace this line with the following if you want to protect against wrapping uints.
    //if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value && balances[_to] + _value > balances[_to]) {
    if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value) {
        balances[_to] += _value;
        balances[_from] -= _value;
        allowed[_from][msg.sender] -= _value;
        emit Transfer(_from, _to, _value);
        return true;
    } else { return false; }
}
```

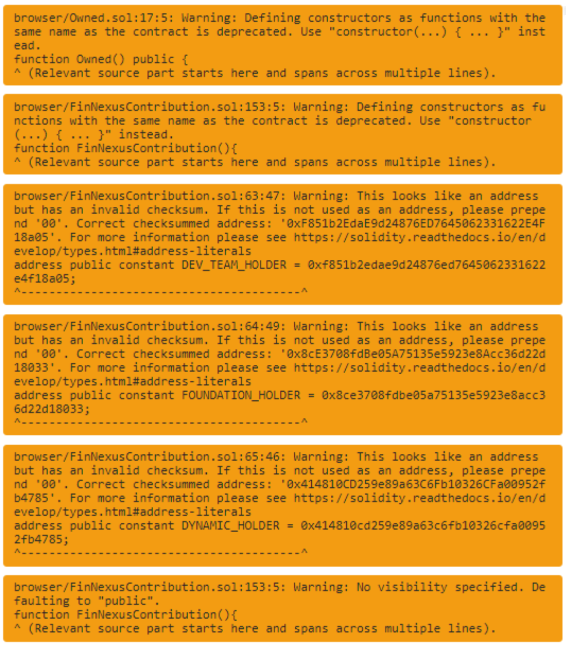
Figure 1 Source code of function ‘transfer’ and ‘transferFrom’

Safety Suggestion: It is recommended to use 'require' check the condition of transferring. It will throw an exception to avoid fake deposit when not satisfied with the condition.

Other audit suggestions:

➢ Compiler warning

Compiled using the 0.4.24 version, there are some contracts have compiler warnings, as shown in Figure 2 is the partial compilation results of file ‘FinNexusContribution.sol’:



```
browser/Owned.sol:17:5: Warning: Defining constructors as functions with the
same name as the contract is deprecated. Use "constructor(...) { ... }" inst
ead.
function Owned() public {
^ (Relevant source part starts here and spans across multiple lines).

browser/FinNexusContribution.sol:153:5: Warning: Defining constructors as fu
nctions with the same name as the contract is deprecated. Use "constructor
(...) { ... }" instead.
function FinNexusContribution(){
^ (Relevant source part starts here and spans across multiple lines).

browser/FinNexusContribution.sol:63:47: Warning: This looks like an address
but has an invalid checksum. If this is not used as an address, please prepe
nd '00'. Correct checksummed address: '0xF851b2EdaE9d24876ED7645062331622E4F
18a05'. For more information please see https://solidity.readthedocs.io/en/d
evelop/types.html#address-literals
address public constant DEV_TEAM_HOLDER = 0xf851b2edae9d24876ed7645062331622
e4f18a05;
^---------------------------------------^

browser/FinNexusContribution.sol:64:49: Warning: This looks like an address
but has an invalid checksum. If this is not used as an address, please prepe
nd '00'. Correct checksummed address: '0x8cE3708fdBe05A75135e5923e8Acc36d22d
18033'. For more information please see https://solidity.readthedocs.io/en/d
evelop/types.html#address-literals
address public constant FOUNDATION_HOLDER = 0x8ce3708fdbe05a75135e5923e8acc3
6d22d18033;
^---------------------------------------^

browser/FinNexusContribution.sol:65:46: Warning: This looks like an address
but has an invalid checksum. If this is not used as an address, please prepe
nd '00'. Correct checksummed address: '0x414810CD259e89a63C6Fb10326CFa00952f
b4785'. For more information please see https://solidity.readthedocs.io/en/d
evelop/types.html#address-literals
address public constant DYNAMIC_HOLDER = 0x414810cd259e89a63c6fb10326cfa0095
2fb4785;
^---------------------------------------^

browser/FinNexusContribution.sol:153:5: Warning: No visibility specified. De
faulting to "public".
function FinNexusContribution(){
^ (Relevant source part starts here and spans across multiple lines).
```

Figure 2 compiler warning screenshot of contract ‘FinNexusContribution’

Safety Suggestion: It is recommended to fix the compiler version and eliminate compiler warnings.

Audited Source Code with Comments:

```
### CfncToken.sol ###
pragma solidity ^0.4.24; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version and
eliminate compiler warnings.

/*

  Copyright 2019 FinNexus Foundation.

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.

*/

import "./StandardToken.sol";
import "./SafeMath.sol";
import "./UM1SToken.sol";

/// @title FinNexus CFNC Token Contract
/// For more information about this token sale, please visit https://FinNexus.org

contract CfncToken is StandardToken {

    using SafeMath for uint; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical
operation. Avoid integer overflow/underflow.

    /// Constant token specific fields
    string public  name = "CfncCoin"; // Beosin (Chengdu LianAn) // The token name is set to "CfncCoin".
    string public  symbol = "CFNC"; // Beosin (Chengdu LianAn) // The token symbol is set to "CFNC".

    uint   public constant decimals = 18; // Beosin (Chengdu LianAn) // The token decimals is set to 18. It's
recommended to use uint8 to declare this variable.

    /// FinNexus total tokens supply
    uint public MAX_TOTAL_TOKEN_AMOUNT = 500000000 ether; // Beosin (Chengdu LianAn) // Declare
the CFNC_TOTAL_SUPPLY, recording the maximum issuance of CFNC token is 500 million.

    uint public constant DIVISOR = 1000;
    /// ERC20 compilant FinNexus UM1S token contact instance
    UM1SToken public um1sToken;

    /// FinNexus contribution contract
    address public minter;  // Beosin (Chengdu LianAn) // Declare the variable 'minter' for storing the address
with mint permission.
    address public initiator; // Beosin (Chengdu LianAn) // Declare the variable 'initiator' for storing the
address with initialization permission.

    /// current phase convert start time
    uint public conStartTime;
    /// current phase convert end time
    uint public conEndTime;

    //the ratio for cfnc which can be changed to UM1S
    uint public conRatio;

    uint public totalSupply; // Beosin (Chengdu LianAn) // Declare the variable 'totalSupply' for storing the
total supply of token.
    uint public totalCfnc2UM1S; // Beosin (Chengdu LianAn) // Declare the variable 'totalCfnc2UM1S' for
storing the total amount of CFNC that has been exchanged for UM1S.
    uint public totalMinted; // Beosin (Chengdu LianAn) // Declare the variable 'totalMinted' for storing the
total amount of minted token.

    /// the firs data for save
    uint public firstPhaseCfnc2UM1S; // Beosin (Chengdu LianAn) // Declare variable 'firstPhaseCfnc2UM1S'
for storing the total amount of CFNC tokens exchanged for UM1S tokens in the first phase.
    uint public firstPhaseTotalSupply; // Beosin (Chengdu LianAn) // Declare variable 'firstPhaseTotalSupply'
for storing the total supply of CFNC tokens in the first phase.

    mapping (address => uint) public phase2Buyer;

    /*
     * EVENTS
     */
    event ConvertCfnc2UM1S(address indexed initiator,uint indexed value);

    event FirstPhaseParameters(uint indexed startTime,uint indexed endTime,uint indexed conRatio);
    event SecondPhaseParameters(uint  indexed startTime,uint  indexed endTime,uint indexed conRatio);

    /*
     * MODIFIERS
     */
    modifier onlyMinter {
        assert(msg.sender == minter);
        _;
    }
    modifier onlyInitiator {
        assert(msg.sender == initiator);
        _;
    }

    // Beosin (Chengdu LianAn) // Modifier, avoid minting exceeding the maximum circulation limit.
    modifier maxWanTokenAmountNotReached (uint amount){
        assert(totalSupply.add(amount) <= MAX_TOTAL_TOKEN_AMOUNT);
        _;
    }

    /**
     * CONSTRUCTOR
     *
     * @dev Initialize the FinNexus Token
     * @param _minter The FinNexus Contribution Contract
     */
    function CfncToken(address _minter,address _initiator){
        minter = _minter; // Beosin (Chengdu LianAn) // Set the address with minting permission to '_minter'.
        initiator = _initiator; // Beosin (Chengdu LianAn) // Set the address with initialization permission to
'_initiator'.

        um1sToken = new UM1SToken(this); // Beosin (Chengdu LianAn) // Generate instance of UM1S token
contract.
    }

    /**
     * func
     *
     * @dev init token contract
     * @param   _phase the phase for mint token
     * @param   _conStartTime the start time for converting cfnc to UM1S
     * @param   _conEndTime the end time for converting cfnc to UM1S
     * @param   _conRatio the conRatio for converting cfnc to UM1S
     */
    function init(uint _phase,uint _conStartTime,uint _conEndTime,uint _conRatio)
        public
        onlyInitiator
    {
       // Beosin (Chengdu LianAn) // Parameter check.
       require(_phase == 1 || _phase == 2);
       require(_conStartTime > 0);
       require(_conEndTime > _conStartTime);
       require(_conRatio > 0);

       conStartTime = _conStartTime; // Beosin (Chengdu LianAn) // The start time of this phase is set to
'_conStartTime'.
```

```
        conRatio = _conRatio; // Beosin (Chengdu LianAn) // Set the conversion ratio.

        if (_phase == 1) {
            conEndTime = _conEndTime; // Beosin (Chengdu LianAn) // Set the end time of the first phase.
            emit FirstPhaseParameters(_conStartTime,_conEndTime,_conRatio); // Beosin (Chengdu LianAn) //
Trigger the event 'FirstPhaseParameters'.
        } else {
            //convert start time for phase 2 must be later than the convert end time for phase 1

            require(_conStartTime > conEndTime);

            conEndTime = _conEndTime; // Beosin (Chengdu LianAn) // Set the end time of the second phase.

            //record the data for 1st stage
            firstPhaseCfnc2UM1S = totalCfnc2UM1S; // Beosin (Chengdu LianAn) // Record the amount of
exchanged token in the first phase.
            totalCfnc2UM1S = 0; // Beosin (Chengdu LianAn) // Reset the token exchange amount at the current
phase to 0.

            firstPhaseTotalSupply = totalMinted; // Beosin (Chengdu LianAn) // Record the total supply of token in
the first phase.

            emit SecondPhaseParameters(_conStartTime,_conEndTime,_conRatio); // Beosin (Chengdu LianAn) //
Trigger the event 'SecondPhaseParameters'.

        }


    }

    /**
     * EXTERNAL FUNCTION
     *
     * @dev mint token for common investor
     * @param _receipent The destination account owned mint tokens
     * @param _amount The amount of mint token be sent to this address.
     *
     */
    function mintToken(address _receipent, uint _amount)
        external
        onlyMinter
        maxWanTokenAmountNotReached(_amount)
    {
        //check parameter in ico minter contract
        balances[_receipent] = balances[_receipent].add(_amount); // Beosin (Chengdu LianAn) // Increase the
token balance of the address '_receipent'.
        totalSupply = totalSupply.add(_amount); // Beosin (Chengdu LianAn) // Update the total supply of token.
        totalMinted = totalMinted.add(_amount); // Beosin (Chengdu LianAn) // Update the total amount of
```

```
minted tokens.

        //phase2 record
        if (firstPhaseTotalSupply > 0) {
            // Beosin (Chengdu LianAn) // Accumulate the number of tokens received at specified address in the
second phase.
            phase2Buyer[_receipent] = phase2Buyer[_receipent].add(_amount);
        }
    }


    /**
     * public FUNCTION
     *
     * @dev convert cfnc to UM1S
     * @param _value The amount converting from cfnc to UM1S
     *
     */
    function convert2UM1S(uint _value)
      public
    {
        require(now >= conStartTime && now <= conEndTime); // Beosin (Chengdu LianAn) // The conversion
time check.

        require(_value >= 0.1 ether); // Beosin (Chengdu LianAn) // The check for the validity of exchange
quantity.
        require(balances[msg.sender] >= _value); // Beosin (Chengdu LianAn) // The balance check.

        if (firstPhaseTotalSupply > 0) {
            require(phase2Buyer[msg.sender] >= _value);
            phase2Buyer[msg.sender] = phase2Buyer[msg.sender].sub(_value);
        }

        //cal quota for convert in current phase,cal it here because we do not know totalSupply until now possible,80%
is allowed to convert
        uint convertQuota  = totalMinted.sub(firstPhaseTotalSupply).mul(conRatio).div(DIVISOR);

        //totalCfnc2UM1S is accumulator for current phase
        uint availble = convertQuota.sub(totalCfnc2UM1S);

        //available token must be over the value converted to UM1S
        assert(availble >= _value);


        balances[msg.sender] = balances[msg.sender].sub(_value); // Beosin (Chengdu LianAn) // Update the token
balance of 'msg.sender'.
        totalCfnc2UM1S = totalCfnc2UM1S.add(_value); // Beosin (Chengdu LianAn) // Update the total amount
of tokens exchanged at the current phase.
```

```
        um1sToken.mintToken(msg.sender, _value); // Beosin (Chengdu LianAn) // Mint UM1S token for 'msg.sender' by calling the function 'mintToken' of UM1S token contract.
        totalSupply = totalSupply.sub(_value); // Beosin (Chengdu LianAn) // Update the total supply of token.

        emit ConvertCfnc2UM1S(msg.sender,_value); // Beosin (Chengdu LianAn) // Trigger the event 'ConvertCfnc2UM1S'.

    }

    /**
     * EXTERNAL FUNCTION
     *
     * @dev change token name
     * @param _name token name
     * @param _symbol token symbol
     *
     */
    function changeTokenName(string _name, string _symbol)
        external
        onlyMinter
    {
        //check parameter in ico minter contract
        name = _name;
        symbol = _symbol;
    }


}


### ERC20Protocol.sol ###
// Abstract contract for the full ERC 20 Token standard
// https://github.com/ethereum/EIPs/issues/20
pragma solidity ^0.4.24; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

contract ERC20Protocol {
    /* This is a slight change to the ERC20 base standard.
    function totalSupply() constant returns (uint supply);
    is replaced with:
    uint public totalSupply;
    This automatically creates a getter function for the totalSupply.
    This is moved to the base contract since public getter functions are not
    currently recognised as an implementation of the matching abstract
    function by the compiler.
    */
    /// total amount of tokens
    uint public totalSupply; // Beosin (Chengdu LianAn) // Declare the variable 'totalSupply' for storing the
```

```
total supply of token.
    // Beosin (Chengdu LianAn) // Define the function interfaces required by the ERC20 Token standard.
    /// @param _owner The address from which the balance will be retrieved
    /// @return The balance
    function balanceOf(address _owner) public constant returns (uint balance);

    /// @notice send `_value` token to `_to` from `msg.sender`
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transfer(address _to, uint _value) public returns (bool success);

    /// @notice send `_value` token to `_to` from `_from` on the condition it is approved by `_from`
    /// @param _from The address of the sender
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transferFrom(address _from, address _to, uint _value) public returns (bool success);

    /// @notice `msg.sender` approves `_spender` to spend `_value` tokens
    /// @param _spender The address of the account able to transfer the tokens
    /// @param _value The amount of tokens to be approved for transfer
    /// @return Whether the approval was successful or not
    function approve(address _spender, uint _value) public returns (bool success);

    /// @param _owner The address of the account owning tokens
    /// @param _spender The address of the account able to transfer the tokens
    /// @return Amount of remaining tokens allowed to spent
    function allowance(address _owner, address _spender) public constant returns (uint remaining);
    // Beosin (Chengdu LianAn) // Define events required by ERC20 Token standard.
    event Transfer(address indexed _from, address indexed _to, uint _value);
    event Approval(address indexed _owner, address indexed _spender, uint _value);
}

### FinNexusContribution.sol ###
pragma solidity ^0.4.24; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version and
eliminate compiler warnings.

/*

  Copyright 2019 FinNexus Foundation.

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0
```

```
 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
*/



import "./SafeMath.sol";
import "./Owned.sol";

/// @title FinNexus Contribution Contract
/// ICO Rules according: https://www.FinNexus.org/crowdsale
/// For more information about this token sale, please visit https://FinNexus.org
// Beosin (Chengdu LianAn) // Declare the function interfaces of the CFNC token contract required by this
contract.
contract CfncTokenInterface {
    function mintToken(address _receipent, uint _amount) external;
    function conEndTime()  public view returns(uint);
}

contract FinNexusContribution is Owned {

    using SafeMath for uint; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical
operation. Avoid integer overflow/underflow.

    /// Constant fields

    /// FinNexus total tokens supply
    uint public constant CFNC_TOTAL_SUPPLY = 500000000 ether;

    /// -----------------------------------------------------------------------------------------------------
    /// |                                        |                  |               |         |
    /// |       PUBLIC SALE (PRESALE + OPEN SALE)         |     DEV TEAM      |   FOUNDATION   |
DYNAMIC |
    /// |                   30%                   |        25%        |       30%       |  15%    |
    /// -----------------------------------------------------------------------------------------------------
    uint public constant OPEN_SALE_STAKE = 300;

    uint public constant FIRST_OPEN_SALE_AMOUNT = 80000000 ether; // Beosin (Chengdu LianAn) // The
total number of tokens used for crowdfunding in the first phase is 80 million.
    uint public constant SECOND_OPEN_SALE_AMOUNT = 70000000 ether; // Beosin (Chengdu LianAn) //
The total number of tokens used for crowdfunding in the second phase is 70 million.

    // Reserved stakes
    uint public constant DEV_TEAM_STAKE = 250;   // 25%
    uint public constant FOUNDATION_STAKE = 300; // 30%
    uint public constant DYNAMIC_STAKE = 150;    // 15%
    uint public constant DIVISOR = 1000;

    /// Exchange rates for WAN
    uint public WAN_CFNC_RATE ;

    /// Addresses of Patrons
    address public constant DEV_TEAM_HOLDER = 0xf851b2edae9d24876ed7645062331622e4f18a05;
    address public constant FOUNDATION_HOLDER = 0x8ce3708fdbe05a75135e5923e8acc36d22d18033;
    address public constant DYNAMIC_HOLDER = 0x414810cd259e89a63c6fb10326cfa00952fb4785;

        ///All deposited wan will be instantly forwarded to this address.
    address public walletAddress;

    ///the pahse for Contribution is divied into 2 phase
    uint public CURRENT_PHASE = 1;

    ///max quota for open sale
    uint public MAX_OPEN_SOLD;

    ///max quota for exhange
    uint public MAX_EXCHANGE_MINT;

    ///contribution start time
    uint public startTime;
    ///contribution end time
    uint public endTime;

    ///accumulator for open sold tokens in current phase
    uint public openSoldTokens;
    ///accumulator for sold tokens in exchange in current phase
    uint public mintExchangeTokens;

    /// Due to an emergency, set this to true to halt the contribution
    bool public halted;

    /// ERC20 compilant FinNexus token contact instance
    address public cfncTokenAddress;

    ///the indicator for initialized status
    bool public isInitialized = false;

    /*
     * EVENTS
     */

    event FirstPhaseTime(uint indexed startTime,uint indexed endTime,uint indexed wanRatioOfSold);
    event SecondPhaseTime(uint indexed startTime,uint indexed endTime,uint indexed wanRatioOfSold);
```

```
    event NewSale(address indexed destAddress, uint indexed wanCost, uint indexed gotTokens);
    event MintExchangeSale(address indexed exchangeAddr, uint indexed amount);

    event contribution(address indexed destAddress, uint indexed wanCost); // Beosin (Chengdu LianAn) //
Unused event, it is recommended to delete.
    /*
     * MODIFIERS
     */
    modifier onlyWallet {
        require(msg.sender == walletAddress);
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, modified functions can only be called during the non-stop phase.
    modifier notHalted() {
        require(!halted);
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, modified functions can only be called after the contract has been
initialized.
    modifier initialized() {
        require(isInitialized);
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, modified functions can only be called at time 'x' and later.
    modifier notEarlierThan(uint x) {
        require(now >= x);
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, modified functions can only be called when the current time is
earlier than time 'x'.
    modifier earlierThan(uint x) {
        require(now < x);
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, modified functions can only be called if the tokens that have been
sold are less than the maximum sale limit.
    modifier ceilingNotReached() {
        require(openSoldTokens < MAX_OPEN_SOLD);
        _;
    }
    // Beosin (Chengdu LianAn) // Unused modifier, it is recommended to delete.
    modifier isSaleEnded() {
        require(now > endTime || openSoldTokens >= MAX_OPEN_SOLD);
        _;
    }
```

```
/**
 * CONSTRUCTOR
 *
 * @dev Initialize the FinNexus contribution contract
 */
function FinNexusContribution(){ // Beosin (Chengdu LianAn) // No operation in constructor, it is 
recommended to delete.

}

/**
 * public function
 *
 * @dev change wallet address for recieving wan
 * @param _walletAddress the address for recieving cfnc tokens
 * @param _cfncTokenAddress the address for cfnc token
 *
 */
function initAddress(address _walletAddress,address _cfncTokenAddress)
    public
    onlyOwner
{
    // Beosin (Chengdu LianAn) // Parameter check.
    require(_walletAddress != 0x0);
    require(_cfncTokenAddress != 0x0);
    // Beosin (Chengdu LianAn) // Initialize wallet address and CFNC token contract address.
    walletAddress = _walletAddress;
    cfncTokenAddress = _cfncTokenAddress;
}


/**
 * public function
 *
 * @dev Initialize the FinNexus contribution contract
 * @param _phase    init the different phase
 * @param _wanRatioOfSold   the ratio for open sale in different phase
 * @param _startTime    start time for open sale
 * @param _endTime  end time for open sale
 * @param _Wan2CfncRate the change rate from wan to cfnc
 *
 */
function init(  uint _phase,
                uint _wanRatioOfSold,
                uint _startTime,
                uint _endTime,
                uint _Wan2CfncRate
                )
```

```
        public
        onlyOwner
    {
        // Beosin (Chengdu LianAn) // Parameter check.
        require(cfncTokenAddress != 0x0 );
        require(_startTime > 0);
        require(_endTime > _startTime);
        require(_Wan2CfncRate > 0);
        // Beosin (Chengdu LianAn) // Initialization phase start & end time and exchange rate of wan to CFNC.
        startTime = _startTime;
        endTime = _endTime;
        WAN_CFNC_RATE =  _Wan2CfncRate;

        if (_phase == 1) {
            // Beosin (Chengdu LianAn) // Distribution of all tokens except crowdfunding.
            /// Reserve tokens according FinNexus ICO rules
            uint stakeMultiplier = CFNC_TOTAL_SUPPLY.div(DIVISOR);
            /// mint tokens for dirrent holder
            CfncTokenInterface(cfncTokenAddress).mintToken(DEV_TEAM_HOLDER,
DEV_TEAM_STAKE.mul(stakeMultiplier));
            CfncTokenInterface(cfncTokenAddress).mintToken(FOUNDATION_HOLDER,
FOUNDATION_STAKE.mul(stakeMultiplier));
            CfncTokenInterface(cfncTokenAddress).mintToken(DYNAMIC_HOLDER,
DYNAMIC_STAKE.mul(stakeMultiplier));
            // Beosin (Chengdu LianAn) // Allocate the total amount of tokens used for crowdfunding and
exchange in the first phase.
            MAX_OPEN_SOLD = FIRST_OPEN_SALE_AMOUNT.mul(_wanRatioOfSold).div(DIVISOR);
            MAX_EXCHANGE_MINT = FIRST_OPEN_SALE_AMOUNT.sub(MAX_OPEN_SOLD);

            emit FirstPhaseTime(_startTime,_endTime,_wanRatioOfSold); // Beosin (Chengdu LianAn) // Trigger
the event 'FirstPhaseTime'.

        } else {
            // Beosin (Chengdu LianAn) // Parameter check.
            require(_phase == 2);
            require(_startTime > CfncTokenInterface(cfncTokenAddress).conEndTime());
            // Beosin (Chengdu LianAn) // Allocate the total amount of tokens used for crowdfunding and
exchange in the second phase.
            MAX_OPEN_SOLD = SECOND_OPEN_SALE_AMOUNT.mul(_wanRatioOfSold).div(DIVISOR);
            MAX_EXCHANGE_MINT = SECOND_OPEN_SALE_AMOUNT.sub(MAX_OPEN_SOLD);

            //initialize variable in the first phase
            mintExchangeTokens = 0;
            openSoldTokens = 0;

            CURRENT_PHASE = 2;

            emit SecondPhaseTime(_startTime,_endTime,_wanRatioOfSold); // Beosin (Chengdu LianAn) // Trigger
the event 'SecondPhaseTime'.
        }

        isInitialized = true; // Beosin (Chengdu LianAn) // The variable 'isInitialized' is set to 'true', indicating
that initialization is complete.
    }

    /**
     * public function
     *
     * @dev minting cfnc tokens for exchange
     * @param _exchangeAddr the exchange address for recieving tokens
     * @param _amount the token amount for exchange
     *
     */
    function mintExchangeToken(address _exchangeAddr, uint _amount)
        public
        notHalted
        initialized
        onlyWallet
    {
        uint availToken = MAX_EXCHANGE_MINT.sub(mintExchangeTokens); // Beosin (Chengdu LianAn) //
Calculate the total number of remaining mintable tokens currently used for exchange.
        assert(availToken > 0);
        // Beosin (Chengdu LianAn) // If the number of tokens required this time does not exceed the total
amount of remaining mintable tokens.
        if (availToken >= _amount) {
            // Beosin (Chengdu LianAn) // Invoke the mintToken function of the CFNC token contract to mint
'_amount' number of tokens to the specified address.
            mintExchangeTokens = mintExchangeTokens.add(_amount);
            CfncTokenInterface(cfncTokenAddress).mintToken(_exchangeAddr, _amount);
            emit MintExchangeSale(_exchangeAddr,_amount);
        } else {
            // Beosin (Chengdu LianAn) // Else, only allowed to mint 'availToken' number of tokens.
            mintExchangeTokens = mintExchangeTokens.add(availToken);
            CfncTokenInterface(cfncTokenAddress).mintToken(_exchangeAddr,availToken);
            emit MintExchangeSale(_exchangeAddr,availToken);
        }
    }


    /**
     * Fallback function
     *
     * @dev If anybody sends Wan directly to this  contract, consider he is getting wan token
     */
    function () public payable {
        buyCfncCoin(msg.sender);
```

```
}

/**
 * public function
 *
 * @dev minting cfnc tokens for exchange
 * @param _receipient the address for recieving cfnc tokens
 *
 */
function buyCfncCoin(address _receipient)
    public
    payable
    notHalted
    ceilingNotReached
    initialized
    notEarlierThan(startTime)
    earlierThan(endTime)
{
    // Beosin (Chengdu LianAn) // Parameter check.
    require(_receipient != 0x0);
    require(msg.value >= 0.1 ether);

    // Do not allow contracts to game the system
    require(tx.origin == msg.sender);

    buyNormal(_receipient); // Beosin (Chengdu LianAn) // Call the internal function 'buyNormal'.
}

/**
 * public function
 * @dev Emergency situation that requires contribution period to stop,Contributing not possible anymore.
 */

function halt() public onlyOwner{
    halted = true;
}

/**
 * public function
 * @dev Emergency situation resolved, Contributing becomes possible again withing the outlined restrictions.
 */

function unHalt() public onlyOwner{
    halted = false;
}

/**
 * public function
```

```
     *
     * @dev set rate from wan to cfnc
     * @param _Wan2CfncRate the exchange rate
     *
     */
    function setExchangeRate(uint _Wan2CfncRate) public onlyOwner{
        require(_Wan2CfncRate != 0);
        WAN_CFNC_RATE =  _Wan2CfncRate; // Beosin (Chengdu LianAn) // The variable
'WAN_CFNC_RATE' is set to '_Wan2CfncRate'.
    }

    /**
     * public function
     *
     * @dev change exchange quota
     * @param _quota the exchange rate
     * @param _add exchange quota direction,increase or decrease
     */
    function changeExchangeQuota(uint _quota,bool _add) public onlyOwner{
        require(_quota != 0);

        if (_add) {
            uint tokenAvailable = MAX_OPEN_SOLD.sub(openSoldTokens); // Beosin (Chengdu LianAn) //
Calculate the maximum amount that can be allocated from the sale quota to the exchange quota.
            // Beosin (Chengdu LianAn) // Adjust quotas based on actual conditions.
            if (tokenAvailable > _quota) {
                MAX_OPEN_SOLD = MAX_OPEN_SOLD.sub(_quota);
                MAX_EXCHANGE_MINT = MAX_EXCHANGE_MINT.add(_quota);
            } else {
                MAX_OPEN_SOLD = MAX_OPEN_SOLD.sub(tokenAvailable);
                MAX_EXCHANGE_MINT = MAX_EXCHANGE_MINT.add(tokenAvailable);
            }
        } else {
            tokenAvailable = MAX_EXCHANGE_MINT.sub(mintExchangeTokens); // Beosin (Chengdu LianAn) //
Calculate the maximum amount that can be allocated from the exchange quota to the sale quota.
            // Beosin (Chengdu LianAn) // Adjust quotas based on actual conditions.
            if (tokenAvailable > _quota) {
                MAX_OPEN_SOLD = MAX_OPEN_SOLD.add(_quota);
                MAX_EXCHANGE_MINT = MAX_EXCHANGE_MINT.sub(_quota);
            } else {
                MAX_OPEN_SOLD = MAX_OPEN_SOLD.add(tokenAvailable);
                MAX_EXCHANGE_MINT = MAX_EXCHANGE_MINT.sub(tokenAvailable);
            }
        }

    }
    /**
     * public function
     *
     * @dev changed wallet address for Emergency
     * @param _newAddress new address
     *
     */
    function changeWalletAddress(address _newAddress) onlyWallet {
        walletAddress = _newAddress; // Beosin (Chengdu LianAn) // Set '_newAddress' as the new wallet
address.
    }

    /////////////// internal function //////////////////////////

    /// @dev Buy FinNexus token normally
    function buyNormal(address receipient) internal {

        // protect partner quota in stage one
        uint tokenAvailable = MAX_OPEN_SOLD.sub(openSoldTokens); // Beosin (Chengdu LianAn) // Calculate
the current remaining token quota available for open sale.
        require(tokenAvailable > 0);

        uint toFund;
        uint toCollect;
        (toFund, toCollect) = costAndBuyTokens(tokenAvailable); // Beosin (Chengdu LianAn) // Call the internal
function 'costAndBuyTokens' to calculate the actual cost of this exchange and the number of exchangeable
tokens.

        buyCommon(receipient, toFund, toCollect); // Beosin (Chengdu LianAn) // Call the internal function
'buyCommon'.
    }

    /// @dev Utility function for bug FinNexus token
    function buyCommon(address receipient, uint toFund, uint tokenCollect) internal {
        require(msg.value >= toFund); // double check

        if(toFund > 0) {
            // Beosin (Chengdu LianAn) // Call the mintToken function of the CFNC token contract to mint
tokens for the specified address and update the record of 'openSoldTokens'.
            CfncTokenInterface(cfncTokenAddress).mintToken(receipient, tokenCollect);
            openSoldTokens = openSoldTokens.add(tokenCollect);

            //transfer wan to specified address
            walletAddress.transfer(toFund);

            emit NewSale(receipient, toFund, tokenCollect); // Beosin (Chengdu LianAn) // Trigger the event
'NewSale'.
        }
        uint toReturn = msg.value.sub(toFund); // Beosin (Chengdu LianAn) // Calculate the remaining amount of
wan after the exchange is completed.
        if(toReturn > 0) {
            msg.sender.transfer(toReturn); // Beosin (Chengdu LianAn) // Return the remaining wan to the
'msg.sender'
        }
    }

    /// @dev Utility function for calculate available tokens and cost wans
    function costAndBuyTokens(uint availableToken) constant internal returns (uint costValue, uint getTokens){

        getTokens = WAN_CFNC_RATE.mul(msg.value).div(DIVISOR); // Beosin (Chengdu LianAn) // Calculate
the number of tokens expected to be exchanged.

        if(availableToken >= getTokens){ // Beosin (Chengdu LianAn) // If the number of tokens expected to be
exchanged is not higher than the remaining quota for sale, set 'costValue' to be 'msg.value'.
            costValue = msg.value;
        } else { // Beosin (Chengdu LianAn) // Else, can only exchange the remaining saleable quota.
            costValue = availableToken.mul(DIVISOR).div(WAN_CFNC_RATE);
            getTokens = availableToken;
        }
    }

}

### Owned.sol ###
pragma solidity ^0.4.24; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

/// @dev `Owned` is a base level contract that assigns an `owner` that can be
///  later changed
contract Owned {

    /// @dev `owner` is the only address that can call a function with this
    /// modifier
    modifier onlyOwner() {
        require(msg.sender == owner); // Beosin (Chengdu LianAn) // Require that the function caller must be
owner.
        _;
    }

    address public owner; // Beosin (Chengdu LianAn) // Declare the variable 'owner' for storing the owner of
contract.

    /// @notice The Constructor assigns the message sender to be `owner`
    function Owned() public { // Beosin (Chengdu LianAn) // Constructor, set owner to the address of deploying
this contract.
        owner = msg.sender;
    }

    address public newOwner; // Beosin (Chengdu LianAn) // Declare the variable 'newOwner' for storing the pending owner of contract.

    /// @notice `owner` can step down and assign some other address to this role
    /// @param _newOwner The address of the new owner. 0x0 can be used to create
    ///  an unowned neutral vault, however that cannot be undone
    function changeOwner(address _newOwner) public onlyOwner {
        newOwner = _newOwner;
    }

    // Beosin (Chengdu LianAn) // The pending owner calls this function to receive ownership.
    function acceptOwnership() public {
        if (msg.sender == newOwner) {
            owner = newOwner;
        }
    }
}

### SafeMath.sol ###
pragma solidity ^0.4.24; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */
library SafeMath {

    /**
    * @dev Multiplies two numbers, reverts on overflow.
    */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
    * @dev Integer division of two numbers truncating the quotient, reverts on division by zero.
    */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0); // Solidity only automatically asserts when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
    * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
    * @dev Adds two numbers, reverts on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
    * @dev Divides two numbers and returns the remainder (unsigned integer modulo),
    * reverts when dividing by zero.
    */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}

### StandardToken.sol ###
/*
You should inherit from StandardToken or, for a token like you would want to
deploy in something like Mist, see HumanStandardToken.sol.
(This implements ONLY the standard functions and NOTHING else.
If you deploy this, you won't have anything useful.)

Implements ERC 20 Token standard: https://github.com/ethereum/EIPs/issues/20
.*/
```

```
pragma solidity ^0.4.24; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version.

import "./ERC20Protocol.sol";
import "./SafeMath.sol";

contract StandardToken is ERC20Protocol {
   using SafeMath for uint; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical
operation. Avoid integer overflow/underflow.

   /**
   * @dev Fix for the ERC20 short address attack.
   */
   modifier onlyPayloadSize(uint size) {
      require(msg.data.length >= size + 4);
      _;
   }
   // Beosin (Chengdu LianAn) // The 'transfer' function, 'msg.sender' transfers the specified amount of
tokens to a specified address.
   function transfer(address _to, uint _value) onlyPayloadSize(2 * 32) public returns (bool success) {
      //Default assumes totalSupply can't be over max (2^256 - 1).
      //If your token leaves out totalSupply and can issue more tokens as time goes on, you need to check if it
doesn't wrap.
      //Replace the if with this one instead.
      //if (balances[msg.sender] >= _value && balances[_to] + _value > balances[_to]) {
      if (balances[msg.sender] >= _value) { // Beosin (Chengdu LianAn) // Balance check of 'msg.sender'.
         // Beosin (Chengdu LianAn) // Update the token balance of both parties and trigger the event
'Transfer'.
         balances[msg.sender] -= _value;
         balances[_to] += _value;
         emit Transfer(msg.sender, _to, _value);
         return true;
      } else { return false; }
   }
   // Beosin (Chengdu LianAn) // The 'transferFrom' function, 'msg.sender' as a delegate of '_from' to
transfer the specified amount of tokens to a specified address.
   function transferFrom(address _from, address _to, uint _value) onlyPayloadSize(3 * 32) public returns (bool
success) {
      //same as above. Replace this line with the following if you want to protect against wrapping uints.
      //if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value && balances[_to] + _value >
balances[_to]) {
      if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value) { // Beosin (Chengdu LianAn) //
Balance and allowance check.
         // Beosin (Chengdu LianAn) // Update the token balance of both parties and decrease the allowance
which 'from' allowed to 'msg.sender' then trigger the event 'Transfer'.
         balances[_to] += _value;
         balances[_from] -= _value;
         allowed[_from][msg.sender] -= _value;
         emit Transfer(_from, _to, _value);
```

```
            return true;
        } else { return false; }
    }
    // Beosin (Chengdu LianAn) // The 'balanceOf' function, returns the token balance of the specified
address.
    function balanceOf(address _owner) public constant returns (uint balance) {
        return balances[_owner];
    }
    // Beosin (Chengdu LianAn) // The 'approve' function, 'msg.sender' allows the specified amount of tokens
to a specified address.
    function approve(address _spender, uint _value) onlyPayloadSize(2 * 32) public returns (bool success) {
        // To change the approve amount you first have to reduce the addresses`
        //  allowance to zero by calling `approve(_spender, 0)` if it is not
        //  already 0 to mitigate the race condition described here:
        //  https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
        assert((_value == 0) || (allowed[msg.sender][_spender] == 0));

        allowed[msg.sender][_spender] = _value; // Beosin (Chengdu LianAn) // The allowance which
'msg.sender' allowed to '_spender' is set to '_value'.
        emit Approval(msg.sender, _spender, _value); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
        return true;
    }
    // Beosin (Chengdu LianAn) // The 'allowance' function, returns the allowance between the specified
addresses.
    function allowance(address _owner, address _spender) public constant returns (uint remaining) {
      return allowed[_owner][_spender];
    }

    mapping (address => uint) balances; // Beosin (Chengdu LianAn) // Declare the mapping variable 'balances'
for storing the token balance of corresponding address.
    mapping (address => mapping (address => uint)) allowed; // Beosin (Chengdu LianAn) // Declare the
mapping variable 'allowed' for storing the allowance between two addresses.
}

### UM1SToken.sol ###
pragma solidity ^0.4.24; // Beosin (Chengdu LianAn) // It is recommended to fix the compiler version and
eliminate compiler warnings.


/*

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.

*/

import "./StandardToken.sol";
import "./SafeMath.sol";


/// @title FinNexus Token Contract
/// For more information about this token sale, please visit https://FinNexus.org
/// @author Cathy - <cathy@FinNexus.org>

contract UM1SToken is StandardToken {

    using SafeMath for uint; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    /// Constant token specific fields
    string public constant name = "UM1SCoin"; // Beosin (Chengdu LianAn) // The token name is set to "UM1SCoin".
    string public constant symbol = "UM1S"; // Beosin (Chengdu LianAn) // The token symbol is set to "UM1S".

    uint public constant decimals = 18; // Beosin (Chengdu LianAn) // The token decimals is set to 18. It's recommended to use uint8 to declare this variable.
    address public minter; // Beosin (Chengdu LianAn) // Declare the variable 'minter' for storing the address with mint permission.

    //5000000000 * 0.8
    uint public constant MAX_TOTAL_TOKEN_AMOUNT = 400000000 ether;

    uint public constant DIVISOR = 1000;
    uint public constant RATE_CFNC_BT = 100;//10:1


    /*
     * MODIFIERS
     */
    modifier onlyMinter {
        assert(msg.sender == minter);
        _;
    }

    /**
     * CONSTRUCTOR
     *
     * @dev Initialize the  BtToken
     */
    // Beosin (Chengdu LianAn) // Constructor, initialize the address of minter to '_minter'.
    function UM1SToken(address _minter){
         minter = _minter;
    }

    /**
     * EXTERNAL FUNCTION
     *
     * @dev mint token for common investor
     * @param _receipent The destination account owned mint tokens
     * @param _amount The amount of mint token be sent to this address.
     *
     */
    function mintToken(address _receipent, uint _amount)
        external
        onlyMinter
    {
       //time check will do in minter cfnc token
        // Beosin (Chengdu LianAn) // Parameter check.
        require(_receipent != 0x0);
        require(_amount > 0);

        uint conAmount = _amount.mul(RATE_CFNC_BT).div(DIVISOR); // Beosin (Chengdu LianAn) // 
Calculate the amount of token that should actually be minted.
        totalSupply = totalSupply.add(conAmount); // Beosin (Chengdu LianAn) // Update the total supply of 
token.

        assert(totalSupply <= MAX_TOTAL_TOKEN_AMOUNT); // Beosin (Chengdu LianAn) // Check the total 
amount of token supply.

        balances[_receipent] = balances[_receipent].add(conAmount); // Beosin (Chengdu LianAn) // Update the 
token balance of the specified address '_receipent'.
    }

}
```

成都链安
BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/LianAnTech