pragma solidity ^0.4.20;

library SafeMath

{

function mul(uint256 a, uint256 b) internal pure returns (uint256)

{

uint256 c = a \* b;

assert(a == 0 || c / a == b);

return c;

}

function div(uint256 a, uint256 b) internal pure returns (uint256)

{

uint256 c = a / b;

return c;

}

function sub(uint256 a, uint256 b) internal pure returns (uint256)

{

assert(b <= a);

return a - b;

}

function add(uint256 a, uint256 b) internal pure returns (uint256)

{

uint256 c = a + b;

assert(c >= a);

return c;

}

}

contract OwnerHelper

{

address public owner;

event OwnerTransferPropose(address indexed \_from, address indexed \_to);

modifier onlyOwner

{

require(msg.sender == owner);

\_;

}

function OwnerHelper() public

{

owner = msg.sender;

}

function transferOwnership(address \_to) onlyOwner public

{

require(\_to != owner);

require(\_to != address(0x0));

owner = \_to;

OwnerTransferPropose(owner, \_to);

}

}

contract ERC20Interface

{

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

event Approval( address indexed \_owner, address indexed \_spender, uint \_value);

function totalSupply() constant public returns (uint \_supply);

function balanceOf( address \_who ) public view returns (uint \_value);

function transfer( address \_to, uint \_value) public returns (bool \_success);

function approve( address \_spender, uint \_value ) public returns (bool \_success);

function allowance( address \_owner, address \_spender ) public view returns (uint \_allowance);

function transferFrom( address \_from, address \_to, uint \_value) public returns (bool \_success);

}

contract JYCToken is ERC20Interface, OwnerHelper

{

using SafeMath for uint;

string public name;

uint public decimals;

string public symbol;

uint public totalSupply;

address public wallet;

uint public maxSupply;

uint public saleSupply;

uint public mktSupply;

uint public dvpSupply;

uint public tokenIssuedSale;

uint public tokenIssuedMkt;

uint public tokenIssuedDvp;

uint public saleEtherReceived;

uint constant private E18 = 1000000000000000000;

uint constant private ethPerToken = 10000;

uint constant private privateSaleBonus = 40;

uint constant private preSaleBonus = 20;

uint constant private crowdSaleBonus = 0;

bool public tokenLock;

uint public privateSaleDate;

uint public privateSaleEndDate;

uint public preSaleDate1;

uint public preSaleEndDate1;

uint public preSaleDate2;

uint public preSaleEndDate2;

uint public crowdSaleDate1;

uint public crowdSaleEndDate1;

uint public crowdSaleDate2;

uint public crowdSaleEndDate2;

mapping (address => uint) internal balances;

mapping (address => mapping ( address => uint )) internal approvals;

mapping (address => bool) internal personalLocks;

mapping (address => uint) internal icoEtherContributeds;

event RemoveLock(address indexed \_who);

event WithdrawMkt(address indexed \_to, uint \_value);

function JYCToken() public

{

name = "JYCToken";

decimals = 18;

symbol = "JYCT";

totalSupply = 200000000 \* E18;

balances[msg.sender] = totalSupply;

owner = msg.sender;

wallet = msg.sender;

maxSupply = 200000000 \* E18;

saleSupply = maxSupply.div(100).mul(60); //60%

mktSupply = maxSupply.div(100).mul(20); //20%

dvpSupply = maxSupply.div(100).mul(20); //20%

tokenIssuedSale = 0;

tokenIssuedMkt = 0;

tokenIssuedDvp = 0;

saleEtherReceived = 0;

tokenLock = true;

privateSaleDate = 1525100400; //1st May

privateSaleEndDate = 1525446000; // 5th May

preSaleDate1 = 1525618800; // 7th

preSaleEndDate1 = 1525791600; //9th

preSaleDate2 = 1525878000; // 10th

preSaleEndDate2 = 1526050800; //12th

crowdSaleDate1 = 1526137200; //13th

crowdSaleEndDate1 = 1526742000; //20th

crowdSaleDate2 = 1526828400; //21th

crowdSaleEndDate2 = 1527433200; //28th

}

function atNow() public constant returns (uint)

{

return now;

}

function () payable public

{

buyCoin();

}

function buyCoin() private

{

}

function isTokenLock(address \_from, address \_to) constant public returns (bool \_success)

{

\_success = false;

if(tokenLock == true)

{

\_success = true;

}

if(personalLocks[\_from] == true || personalLocks[\_to] == true)

{

\_success = true;

}

return \_success;

}

function isPersonalLock(address \_who) constant public returns (bool)

{

return personalLocks[\_who];

}

function removeTokenLock() onlyOwner public

{

require(tokenLock == true);

tokenLock = false;

RemoveLock(0x0);

}

function removePersonalTokenLock(address \_person) onlyOwner public

{

require(personalLocks[\_person] == true);

personalLocks[\_person] = false;

RemoveLock(\_person);

}

function totalSupply() constant public returns (uint)

{

return totalSupply;

}

function balanceOf(address \_who) public view returns (uint)

{

return balances[\_who];

}

function transfer(address \_to, uint \_value) public returns (bool)

{

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

require(isTokenLock(msg.sender,\_to) == true);

balances[msg.sender] = balances[msg.sender].sub(\_value);

balances[\_to] = balances[\_to].add(\_value);

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

return true;

}

function approve(address \_spender, uint \_value) public returns (bool)

{

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

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

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

return true;

}

function allowance(address \_owner, address \_spender) constant public returns (uint)

{

return approvals[\_owner][\_spender];

}

function transferFrom(address \_from, address \_to, uint \_value) public returns (bool)

{

require(balances[\_from] >= \_value);

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

require(isTokenLock(\_from,\_to) == true);

approvals[\_from][msg.sender] = approvals[\_from][msg.sender].sub(\_value);

balances[\_from] = balances[\_from].sub(\_value);

balances[\_to] = balances[\_to].add(\_value);

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

return true;

}

function withdrawMktTokens(address \_to, uint \_value) onlyOwner public returns (bool)

{

require(mktSupply > tokenIssuedMkt);

require(mktSupply > tokenIssuedMkt.add(\_value));

balances[\_to] = balances[\_to].add(\_value);

WithdrawMkt(\_to, \_value);

return true;

}

}