|  |
| --- |
| pragma solidity ^0.4.24; |
|  |  |  |
|  |  | library SafeMath { |
|  |  | function mul(uint256 a, uint256 b) internal pure returns (uint256) { |
|  |  | if (a == 0) { |
|  |  | return 0; |
|  |  | } |
|  |  | uint256 c = a \* b; |
|  |  | assert(c / a == b); |
|  |  | return c; |
|  |  | } |
|  |  |  |
|  |  | function div(uint256 a, uint256 b) internal pure returns (uint256) { |
|  |  | // assert(b > 0); // Solidity automatically throws 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; |
|  |  | } |
|  |  |  |
|  |  | 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 Ownable { |
|  |  | address public owner; |
|  |  |  |
|  |  |  |
|  |  | event OwnershipTransferred(address indexed previousOwner, address indexed newOwner); |
|  |  |  |
|  |  | /\*\* |
|  |  | \* @dev Throws if called by any account other than the owner. |
|  |  | \*/ |
|  |  | modifier onlyOwner() { |
|  |  | require(msg.sender == owner); |
|  |  | \_; |
|  |  | } |
|  |  |  |
|  |  |  |
|  |  | /\*\* |
|  |  | \* @dev Allows the current owner to transfer control of the contract to a newOwner. |
|  |  | \* @param newOwner The address to transfer ownership to. |
|  |  | \*/ |
|  |  | function transferOwnership(address newOwner) public onlyOwner { |
|  |  | require(newOwner != address(0)); |
|  |  | emit OwnershipTransferred(owner, newOwner); |
|  |  | owner = newOwner; |
|  |  | } |
|  |  |  |
|  |  | } |
|  |  |  |
|  |  | contract Pausable is Ownable { |
|  |  | event Pause(); |
|  |  | event Unpause(); |
|  |  |  |
|  |  | bool public paused = false; |
|  |  |  |
|  |  |  |
|  |  | /\*\* |
|  |  | \* @dev Modifier to make a function callable only when the contract is not paused. |
|  |  | \*/ |
|  |  | modifier whenNotPaused() { |
|  |  | require(!paused); |
|  |  | \_; |
|  |  | } |
|  |  |  |
|  |  | /\*\* |
|  |  | \* @dev Modifier to make a function callable only when the contract is paused. |
|  |  | \*/ |
|  |  | modifier whenPaused() { |
|  |  | require(paused); |
|  |  | \_; |
|  |  | } |
|  |  |  |
|  |  | /\*\* |
|  |  | \* @dev called by the owner to pause, triggers stopped state |
|  |  | \*/ |
|  |  | function pause() onlyOwner whenNotPaused public { |
|  |  | paused = true; |
|  |  | emit Pause(); |
|  |  | } |
|  |  |  |
|  |  | /\*\* |
|  |  | \* @dev called by the owner to unpause, returns to normal state |
|  |  | \*/ |
|  |  | function unpause() onlyOwner whenPaused public { |
|  |  | paused = false; |
|  |  | emit Unpause(); |
|  |  | } |
|  |  | } |
|  |  |  |
|  |  | contract ERC20Basic { |
|  |  | uint256 public totalSupply; |
|  |  | function balanceOf(address who) public view returns (uint256); |
|  |  | function transfer(address to, uint256 value) public returns (bool); |
|  |  | event Transfer(address indexed from, address indexed to, uint256 value); |
|  |  | } |
|  |  |  |
|  |  | contract ERC20 is ERC20Basic { |
|  |  | function allowance(address owner, address spender) public view returns (uint256); |
|  |  | function transferFrom(address from, address to, uint256 value) public returns (bool); |
|  |  | function approve(address spender, uint256 value) public returns (bool); |
|  |  | event Approval(address indexed owner, address indexed spender, uint256 value); |
|  |  | } |
|  |  |  |
|  |  |  |
|  |  | contract StandardToken is ERC20 { |
|  |  | using SafeMath for uint256; |
|  |  |  |
|  |  | mapping (address => mapping (address => uint256)) internal allowed; |
|  |  | mapping(address => bool) tokenBlacklist; |
|  |  | event Blacklist(address indexed blackListed, bool value); |
|  |  |  |
|  |  |  |
|  |  | mapping(address => uint256) balances; |
|  |  |  |
|  |  |  |
|  |  | function transfer(address \_to, uint256 \_value) public returns (bool) { |
|  |  | require(tokenBlacklist[msg.sender] == false); |
|  |  | require(\_to != address(0)); |
|  |  | require(\_value <= balances[msg.sender]); |
|  |  |  |
|  |  | // SafeMath.sub will throw if there is not enough balance. |
|  |  | balances[msg.sender] = balances[msg.sender].sub(\_value); |
|  |  | balances[\_to] = balances[\_to].add(\_value); |
|  |  | emit Transfer(msg.sender, \_to, \_value); |
|  |  | return true; |
|  |  | } |
|  |  |  |
|  |  |  |
|  |  | function balanceOf(address \_owner) public view returns (uint256 balance) { |
|  |  | return balances[\_owner]; |
|  |  | } |
|  |  |  |
|  |  | function transferFrom(address \_from, address \_to, uint256 \_value) public returns (bool) { |
|  |  | require(tokenBlacklist[msg.sender] == false); |
|  |  | require(\_to != address(0)); |
|  |  | require(\_value <= balances[\_from]); |
|  |  | require(\_value <= allowed[\_from][msg.sender]); |
|  |  |  |
|  |  | balances[\_from] = balances[\_from].sub(\_value); |
|  |  | balances[\_to] = balances[\_to].add(\_value); |
|  |  | allowed[\_from][msg.sender] = allowed[\_from][msg.sender].sub(\_value); |
|  |  | emit Transfer(\_from, \_to, \_value); |
|  |  | return true; |
|  |  | } |
|  |  |  |
|  |  |  |
|  |  | function approve(address \_spender, uint256 \_value) public returns (bool) { |
|  |  | allowed[msg.sender][\_spender] = \_value; |
|  |  | emit Approval(msg.sender, \_spender, \_value); |
|  |  | return true; |
|  |  | } |
|  |  |  |
|  |  |  |
|  |  | function allowance(address \_owner, address \_spender) public view returns (uint256) { |
|  |  | return allowed[\_owner][\_spender]; |
|  |  | } |
|  |  |  |
|  |  |  |
|  |  | function increaseApproval(address \_spender, uint \_addedValue) public returns (bool) { |
|  |  | allowed[msg.sender][\_spender] = allowed[msg.sender][\_spender].add(\_addedValue); |
|  |  | emit Approval(msg.sender, \_spender, allowed[msg.sender][\_spender]); |
|  |  | return true; |
|  |  | } |
|  |  |  |
|  |  | function decreaseApproval(address \_spender, uint \_subtractedValue) public returns (bool) { |
|  |  | uint oldValue = allowed[msg.sender][\_spender]; |
|  |  | if (\_subtractedValue > oldValue) { |
|  |  | allowed[msg.sender][\_spender] = 0; |
|  |  | } else { |
|  |  | allowed[msg.sender][\_spender] = oldValue.sub(\_subtractedValue); |
|  |  | } |
|  |  | emit Approval(msg.sender, \_spender, allowed[msg.sender][\_spender]); |
|  |  | return true; |
|  |  | } |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | function \_blackList(address \_address, bool \_isBlackListed) internal returns (bool) { |
|  |  | require(tokenBlacklist[\_address] != \_isBlackListed); |
|  |  | tokenBlacklist[\_address] = \_isBlackListed; |
|  |  | emit Blacklist(\_address, \_isBlackListed); |
|  |  | return true; |
|  |  | } |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | } |
|  |  |  |
|  |  | contract PausableToken is StandardToken, Pausable { |
|  |  |  |
|  |  | function transfer(address \_to, uint256 \_value) public whenNotPaused returns (bool) { |
|  |  | return super.transfer(\_to, \_value); |
|  |  | } |
|  |  |  |
|  |  | function transferFrom(address \_from, address \_to, uint256 \_value) public whenNotPaused returns (bool) { |
|  |  | return super.transferFrom(\_from, \_to, \_value); |
|  |  | } |
|  |  |  |
|  |  | function approve(address \_spender, uint256 \_value) public whenNotPaused returns (bool) { |
|  |  | return super.approve(\_spender, \_value); |
|  |  | } |
|  |  |  |
|  |  | function increaseApproval(address \_spender, uint \_addedValue) public whenNotPaused returns (bool success) { |
|  |  | return super.increaseApproval(\_spender, \_addedValue); |
|  |  | } |
|  |  |  |
|  |  | function decreaseApproval(address \_spender, uint \_subtractedValue) public whenNotPaused returns (bool success) { |
|  |  | return super.decreaseApproval(\_spender, \_subtractedValue); |
|  |  | } |
|  |  |  |
|  |  | function blackListAddress(address listAddress, bool isBlackListed) public whenNotPaused onlyOwner returns (bool success) { |
|  |  | return super.\_blackList(listAddress, isBlackListed); |
|  |  | } |
|  |  |  |
|  |  | } |
|  |  |  |
|  |  | contract WEALTHMART is PausableToken { |
|  |  | string public name; |
|  |  | string public symbol; |
|  |  | uint public decimals; |
|  |  | event Mint(address indexed from, address indexed to, uint256 value); |
|  |  | event Burn(address indexed burner, uint256 value); |
|  |  |  |
|  |  |  |
|  |  | constructor(string memory \_name, string memory \_symbol, uint256 \_decimals, uint256 \_supply, address tokenOwner) public { |
|  |  | name = \_name; |
|  |  | symbol = \_symbol; |
|  |  | decimals = \_decimals; |
|  |  | totalSupply = \_supply \* 10\*\*\_decimals; |
|  |  | balances[tokenOwner] = totalSupply; |
|  |  | owner = tokenOwner; |
|  |  | emit Transfer(address(0), tokenOwner, totalSupply); |
|  |  | } |
|  |  |  |
|  |  | function burn(uint256 \_value) public { |
|  |  | \_burn(msg.sender, \_value); |
|  |  | } |
|  |  |  |
|  |  | function \_burn(address \_who, uint256 \_value) internal { |
|  |  | require(\_value <= balances[\_who]); |
|  |  | balances[\_who] = balances[\_who].sub(\_value); |
|  |  | totalSupply = totalSupply.sub(\_value); |
|  |  | emit Burn(\_who, \_value); |
|  |  | emit Transfer(\_who, address(0), \_value); |
|  |  | } |
|  |  |  |
|  |  | function mint(address account, uint256 amount) onlyOwner public { |
|  |  |  |
|  |  | totalSupply = totalSupply.add(amount); |
|  |  | balances[account] = balances[account].add(amount); |
|  |  | emit Mint(address(0), account, amount); |
|  |  | emit Transfer(address(0), account, amount); |
|  |  | } |
|  |  |  |
|  |  |  |
|  |  | } |