SOLARTOKEN Smartcontract

CODE:

```
pragma solidity ^0.5.1;
/**
* @title ERC20Basic
* @dev Simpler version of ERC20 interface
* @dev see https://github.com/ethereum/EIPs/issues/179
*/
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);
}
/**
* @title ERC20 interface
* @dev see https://github.com/ethereum/EIPs/issues/20
*/
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);
```

```
}
/**
* @title SafeMath
* @dev Math operations with safety checks that throw on error
*/
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) {
37
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;
}
}
/**
* @title Basic token
* @dev Basic version of StandardToken, with no allowances.
*/
contract BasicToken is ERC20Basic {
using SafeMath for uint256;
mapping(address => uint256) balances;
/**
* @dev transfer token for a specified address
* @param _to The address to transfer to.
* @param _value The amount to be transferred.
*/
function transfer(address _to, uint256 _value) public returns (bool) {
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;
}
/**
* @dev Gets the balance of the specified address.
* @param _owner The address to query the the balance of.
* @return An uint256 representing the amount owned by the passed
address.
*/
function balanceOf(address _owner) public view returns (uint256 balance) {
return balances[_owner];
}
}
/**
* @title Standard ERC20 token
*
* @dev Implementation of the basic standard token.
* @dev https://github.com/ethereum/EIPs/issues/20
* @dev Based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBlo
odToken.sol
*/
38
contract StandardToken is ERC20, BasicToken {
mapping (address => mapping (address => uint256)) internal allowed;
/**
* @dev Transfer tokens from one address to another
```

```
* @param _from address The address which you want to send tokens from
* @param _to address The address which you want to transfer to
* @param _value uint256 the amount of tokens to be transferred
*/
function transferFrom(address _from, address _to, uint256 _value) public
returns (bool) {
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;
}
/**
* @dev Approve the passed address to spend the specified amount of tokens
on behalf of msg.sender.
*
* 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
```

```
* @param _spender The address which will spend the funds.
* @param _value The amount of tokens to be spent.
*/
function approve(address _spender, uint256 _value) public returns (bool) {
allowed[msg.sender][_spender] = _value;
emit Approval(msg.sender, _spender, _value);
return true;
}
/**
* @dev Function to check the amount of tokens that an owner allowed to a
spender.
* @param _owner address The address which owns the funds.
* @param _spender address The address which will spend the funds.
* @return A uint256 specifying the amount of tokens still available for the
spender.
*/
function allowance(address _owner, address _spender) public view returns
(uint256) {
return allowed[_owner][_spender];
}
/**
* approve should be called when allowed[_spender] == 0. To increment
* allowed value is better to use this function to avoid 2 calls (and wait until
39
* the first transaction is mined)
* From MonolithDAO Token.sol
```

```
*/
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;
}
}
/**
* @title Ownable
* @dev The Ownable contract has an owner address, and provides basic
authorization control
* functions, this simplifies the implementation of "user permissions".
*/
```

```
contract Ownable {
address public owner;
event OwnershipTransferred(address indexed previousOwner, address
indexed newOwner);
/**
* @dev The Ownable constructor sets the original `owner` of the contract to
the sender
* account.
*/
constructor () public {
owner = msg.sender;
}
/**
* @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.
*/
40
function transferOwnership(address newOwner) public onlyOwner {
```

```
require(newOwner != address(0));
emit OwnershipTransferred(owner, newOwner);
owner = newOwner;
}
}
/**
* @title Pausable
* @dev Base contract which allows children to implement an emergency stop
mechanism.
*/
contract Pausable is Ownable {
event PausePublic(bool newState);
event PauseOwnerAdmin(bool newState);
bool public pausedPublic = true;
bool public pausedOwnerAdmin = false;
address public admin;
/**
* @dev Modifier to make a function callable based on pause states.
*/
modifier whenNotPaused() {
if(pausedPublic) {
if(!pausedOwnerAdmin) {
require(msg.sender == admin || msg.sender == owner);
} else {
revert();
```

```
}
}
_;
}
/**
* @dev called by the owner to set new pause flags
* pausedPublic can't be false while pausedOwnerAdmin is true
*/
function pause(bool newPausedPublic, bool newPausedOwnerAdmin)
onlyOwner public {
require(!(newPausedPublic == false && newPausedOwnerAdmin == true));
pausedPublic = newPausedPublic;
pausedOwnerAdmin = newPausedOwnerAdmin;
emit PausePublic(newPausedPublic);
emit PauseOwnerAdmin(newPausedOwnerAdmin);
}
}
contract PausableToken is StandardToken, Pausable {
function transfer(address _to, uint256 _value) public whenNotPaused returns
(bool) {
return super.transfer(_to, _value);
}
41
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);
}
}
contract SolarToken is PausableToken {
string public constant name = "Solar";
string public constant symbol = "STR";
uint8 public constant decimals = 4;
modifier validDestination( address to )
{
require(to != address(0x0));
require(to != address(this));
_;
}
constructor ( address _admin, uint _totalTokenAmount ) public
{
```

```
// assign the admin account
admin = _admin;
// assign the total tokens to Solar
totalSupply = _totalTokenAmount;
balances[msg.sender] = _totalTokenAmount;
emit Transfer(address(0x0), msg.sender, _totalTokenAmount);
}
function transfer(address _to, uint _value) validDestination(_to) public
returns (bool)
{
return super.transfer(_to, _value);
}
function transferFrom(address _from, address _to, uint _value)
validDestination(_to) public returns (bool)
{
return super.transferFrom(_from, _to, _value);
}
event Burn(address indexed _burner, uint _value);
42
function burn(uint _value) public returns (bool)
{
balances[msg.sender] = balances[msg.sender].sub(_value);
totalSupply = totalSupply.sub(_value);
emit Burn(msg.sender, _value);
emit Transfer(msg.sender, address(0x0), _value);
```

```
return true;
}
// save some gas by making only one contract call
function burnFrom(address _from, uint256 _value) public returns (bool)
{
assert( transferFrom( _from, msg.sender, _value ) );
return burn(_value);
}
function emergencyERC20Drain( ERC20 token, uint amount ) public
onlyOwner {
// owner can drain tokens that are sent here by mistake
token.transfer( owner, amount );
}
event AdminTransferred(address indexed previousAdmin, address indexed
newAdmin);
function changeAdmin(address newAdmin) public onlyOwner {
// owner can re-assign the admin
emit AdminTransferred(admin, newAdmin);
admin = newAdmin;
}
}
```