```
/**
 *Submitted for verification at polygonscan.com on 2022-10-07
*/

/**
 *Submitted for verification at polygonscan.com on 2022-08-23
*/

/**
 *Submitted for verification at polygonscan.com on 2022-07-30
*/

//SPDX-License-Identifier: MIT

pragma solidity ^0.6.8;

interface IERC20 {
    function balanceOf(address who) external view returns (uint256);
    function transfer(address to, uint256 value) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function transferFrom(address from, address to, uint256 value) external returns (bool);
    function approve(address spender, uint256 value) external returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

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 private _owner;
  constructor() public {
    _owner = msg.sender;
  }

  modifier onlyOwner() {
    require(isOwner(), "Ownable: caller is not the owner");
    _;
  }

  function owner(
  ) public view returns (address) {
    return _owner;
  }

  function isOwner(
  ) public view returns (bool) {
    return msg.sender == _owner;
  }
}

contract Pausable is Ownable {

    event Paused(address account);
    event Unpaused(address account);

    bool private _paused;

    modifier whenNotPaused()
    {
        require(!_paused, "Pausable: paused");
        _;
    }

    modifier whenPaused() {
        require(_paused, "Pausable: not paused");
        _;
    }

    constructor() internal {}

    function paused(
    ) public view returns (bool)
    {
        return _paused;
    }
```

```
    function pause(
    ) public
        onlyOwner
        whenNotPaused
    {
        _paused = true;
        emit Paused(msg.sender);
    }

    function unpause(
    ) public
        onlyOwner
        whenPaused
    {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}

contract ERC20 is IERC20, Ownable, Pausable {

    using SafeMath for uint;

    string internal _name;
    string internal _symbol;
    uint8 internal _decimals;
    uint256 internal _totalSupply;

    mapping (address => uint256) internal _balances;
    mapping (address => mapping (address => uint256)) internal _allowed;
    mapping(address => uint256) internal InvestorSpent;
    event Mint(address indexed minter, address indexed account, uint256 amount);
    event Burn(address indexed burner, address indexed account, uint256 amount);

    uint InitBlock;
    uint FinalBlock;
    uint PInitialBlock;
    uint Pspent;


    address  internal investorAddress;
    address internal CommunityEcosystem;
    address internal Team;
    address internal Development_and_marketing;
    address internal Private_sale;
    address internal Airdrop;
    address internal advisory;

    constructor (


    ) public
```

```
{
    _symbol = 'LST';
    _name = 'Lemon Social';
    _decimals = 18;
    _totalSupply = 100000000000000000000000000000;
     _balances[msg.sender] = 100000000000000000000000000000;



}


//investors contract
function checkSpentRate(uint256 _am, address sender) internal view returns (bool){
    uint256 totalSpent = InvestorSpent[sender] + _am;
    uint duration = (FinalBlock - block.timestamp) / 2592000;
    uint allowed = duration * 850000000000000000000000000;
    if(totalSpent > allowed){
        return false;
    }else{
        return true;
    }
}

function checkInvestor(uint amount, address investor) internal view returns (bool){
    if(InitBlock > block.timestamp){
        return false;
    }else if(InitBlock < FinalBlock ){
        // require(, 'Spending limit exceded');
        if(!checkSpentRate(amount, investor)){
            return false;
        }else{
            return true;
        }
    }else{
        return true;
    }
}

//end of investors contract



 //Private Sale contract
function PcheckSpentRate(uint256 _am) internal view returns (bool){
    uint256 totalSpent = Pspent + _am;
    uint duration = (PInitialBlock - block.timestamp) / 2592000;
    uint allowed = duration * 285000000000000000000000000;
    if(totalSpent > allowed){
        return false;
    }else{
        return true;
```

```
    }
}

function PrivateSaleChecker(uint amount) internal view returns (bool){
  if(PInitialBlock > block.timestamp ){
        // require(, 'Spending limit exceeded');
        if(!PcheckSpentRate(amount)){
            return false;
        }else{
            return true;
        }
    }else{
        return true;
    }
}
//end of Private State contract




function name(
) public view returns (string memory)
{
    return _name;
}

function symbol(
) public view returns (string memory)
{
    return _symbol;
}

function decimals(
) public view returns (uint8)
{
    return _decimals;
}

function totalSupply(
) public view returns (uint256)
{
    return _totalSupply;
}

function transfer(
    address _to,
    uint256 _value
) public override
    whenNotPaused
  returns (bool)
{
```

```
    require(_to != address(0), 'ERC20: to address is not valid');
    require(_value <= _balances[msg.sender], 'ERC20: insufficient balance');

    if(msg.sender == Private_sale){
       require(PrivateSaleChecker(_value), 'Limit exceded');
       Pspent += _value;
    }
    if(msg.sender == investorAddress){
        require(checkInvestor(_value, msg.sender), 'Limit exceded');
        InvestorSpent[msg.sender] += _value;

    }
    _balances[msg.sender] = SafeMath.sub(_balances[msg.sender], _value);
    _balances[_to] = SafeMath.add(_balances[_to], _value);

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

    return true;
 }

function balanceOf(
   address _owner
 ) public override view returns (uint256 balance)
 {
    return _balances[_owner];
 }

 function approve(
   address _spender,
   uint256 _value
 ) public override
    whenNotPaused
  returns (bool)
 {
    _allowed[msg.sender][_spender] = _value;

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

    return true;
}

function transferFrom(
    address _from,
    address _to,
    uint256 _value
 ) public override
    whenNotPaused
  returns (bool)
 {
    require(_from != address(0), 'ERC20: from address is not valid');
    require(_to != address(0), 'ERC20: to address is not valid');
    require(_value <= _balances[_from], 'ERC20: insufficient balance');
```

```
        require(_value <= _allowed[_from][msg.sender], 'ERC20: from not allowed');

        _balances[_from] = SafeMath.sub(_balances[_from], _value);
        _balances[_to] = SafeMath.add(_balances[_to], _value);
        _allowed[_from][msg.sender] = SafeMath.sub(_allowed[_from][msg.sender], _value);

        emit Transfer(_from, _to, _value);

        return true;
    }

    function allowance(
        address _owner,
        address _spender
    ) public override view
        whenNotPaused
      returns (uint256)
    {
        return _allowed[_owner][_spender];
    }

    function increaseApproval(
        address _spender,
        uint _addedValue
    ) public
        whenNotPaused
      returns (bool)
    {
        _allowed[msg.sender][_spender] = SafeMath.add(_allowed[msg.sender][_spender],
_addedValue);

        emit Approval(msg.sender, _spender, _allowed[msg.sender][_spender]);

        return true;
    }

    function decreaseApproval(
        address _spender,
        uint _subtractedValue
    ) public
        whenNotPaused
      returns (bool)
    {
        uint oldValue = _allowed[msg.sender][_spender];

        if (_subtractedValue > oldValue) {
            _allowed[msg.sender][_spender] = 0;
        } else {
            _allowed[msg.sender][_spender] = SafeMath.sub(oldValue, _subtractedValue);
        }

        emit Approval(msg.sender, _spender, _allowed[msg.sender][_spender]);
```

```
        return true;
    }

    function mintTo(
        address _to,
        uint _amount
    ) public
        whenNotPaused
        onlyOwner
    {
        require(_to != address(0), 'ERC20: to address is not valid');
        require(_amount > 0, 'ERC20: amount is not valid');

        _totalSupply = _totalSupply.add(_amount);
        _balances[_to] = _balances[_to].add(_amount);

        emit Mint(msg.sender, _to, _amount);
    }

    function burnFrom(
        address _from,
        uint _amount
    ) public
        whenNotPaused
        onlyOwner
    {
        require(_from != address(0), 'ERC20: from address is not valid');
        require(_balances[_from] >= _amount, 'ERC20: insufficient balance');

        _balances[_from] = _balances[_from].sub(_amount);
        _totalSupply = _totalSupply.sub(_amount);

        emit Burn(msg.sender, _from, _amount);
    }

}
```