/\*\*

\*Submitted for verification at BscScan.com on 2021-02-22

\*/

/\*\*

\*Submitted for verification at BscScan.com on 2021-02-21

\*/

pragma solidity ^0.5.0;

interface IBEP20 {

function totalSupply() external view returns (uint256);

function balanceOf(address who) external view returns (uint256);

function allowance(address owner, address spender) external view returns (uint256);

function transfer(address to, uint256 value) external returns (bool);

function approve(address spender, uint256 value) external returns (bool);

function transferFrom(address from, address to, 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);

event AddedWhiteList(address \_user);

event RemovedWhiteList(address \_user);

event Pause();

event Unpause();

}

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) {

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;

}

function ceil(uint256 a, uint256 m) internal pure returns (uint256) {

uint256 c = add(a,m);

uint256 d = sub(c,1);

return mul(div(d,m),m);

}

}

contract BEP20Detailed is IBEP20 {

string private \_name;

string private \_symbol;

uint8 private \_decimals;

constructor(string memory name, string memory symbol, uint8 decimals) public {

\_name = name;

\_symbol = symbol;

\_decimals = decimals;

}

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;

}

}

contract GST is BEP20Detailed {

using SafeMath for uint256;

mapping (address => uint256) private \_balances;

mapping (address => mapping (address => uint256)) private \_allowed;

mapping (address => bool) public isWhiteListed;

string constant tokenName = "Gemstone Token";

string constant tokenSymbol = "GST";

uint8 constant tokenDecimals = 18;

uint256 \_totalSupply = 15000000\*10\*\*18;

uint256 \_exchangeSupply = 5000000\*10\*\*18;

uint256 \_burnSupply = 2500000\*10\*\*18;

uint256 \_burnPerMonth = 150000\*10\*\*18;

uint256 \_marketingSupply = 2250000\*10\*\*18;

uint256 \_airdropSupply = 1500000\*10\*\*18;

uint256 \_teamSupply = 3750000\*10\*\*18;

uint256 \_releasePerMonth = 1250000\*10\*\*18;

uint256 public basePercent = 5;

uint256 public totalBurn = 0;

uint256 public totalRelease = 0;

uint256 public toBurn = 1000000\*10\*\*18;

uint256 public transferLimit = 150000\*10\*\*18;

uint256 public maxBurnLimit = 1500000\*10\*\*18;

uint256 public monthlyBurn = 0;

uint256 public contractTime = block.timestamp;

uint256 public releaseStartTime = block.timestamp+90 days;

uint256 public nextBurnTime = block.timestamp+30 days;

bool public limitPaused = false;

address payable public exchangeAddress = 0x6A8b40A92BEa9ab666e02DDf7a81a6AF7369CC74;

address payable public burnAddress = 0x3757486FE86Dd8D21f5aCAcA4e864c69b5F2aca8;

address payable public marketingAddress = 0x14401Ea8aCc28da8b7BcCdeBBa4f1A5ee8aF0328;

address payable public airdropAddress = 0xb32Ba22e21bE47a3558Aa836415D9A9E6F2C75D6;

address payable public teamAddress = 0xFdc60D1652BfA0333977740E1697A57a04Af1E6F;

constructor() public payable BEP20Detailed(tokenName, tokenSymbol, tokenDecimals) {

\_mint(exchangeAddress, \_exchangeSupply);

\_mint(burnAddress, \_burnSupply);

\_mint(marketingAddress, \_marketingSupply);

\_mint(airdropAddress, \_airdropSupply);

\_mint(teamAddress, \_teamSupply);

}

function totalSupply() public view returns (uint256) {

return \_totalSupply;

}

function balanceOf(address owner) public view returns (uint256) {

return \_balances[owner];

}

function getLiquidityStatus(address \_maker) public view returns (bool) {

return isWhiteListed[\_maker];

}

function addLiquidityAddress(address \_liquidityUser) public {

require(msg.sender == exchangeAddress, "exchange address not found");

isWhiteListed[\_liquidityUser] = true;

emit AddedWhiteList(\_liquidityUser);

}

function removeLiquidityAddress (address \_liquidityUser) public {

require(msg.sender == exchangeAddress, "exchange address not found");

isWhiteListed[\_liquidityUser] = false;

emit RemovedWhiteList(\_liquidityUser);

}

modifier whenNotPaused() {

require(!limitPaused);

\_;

}

modifier whenPaused() {

require(limitPaused);

\_;

}

function limitoff() whenNotPaused public {

require(msg.sender == exchangeAddress, "exchange address not found");

limitPaused = true;

emit Pause();

}

function limiton() whenPaused public {

require(msg.sender == exchangeAddress, "exchange address not found");

limitPaused = false;

emit Unpause();

}

function findFivePercent(uint256 value) public view returns (uint256){

uint256 roundValue = value.ceil(basePercent);

uint256 fivePercent = roundValue.mul(basePercent).div(100);

return fivePercent;

}

function allowance(address owner, address spender) public view returns (uint256){

return \_allowed[owner][spender];

}

function transfer(address to, uint256 value) public returns (bool)

{

require(value <= \_balances[msg.sender], "transfer amount exceeds balance");

require(to != address(0), "can't transfer to the zero address");

require(to != burnAddress, "can't transfer to the burn address");

require(msg.sender != burnAddress, "can't transfer from the burn address");

require(to != teamAddress, "can't transfer to the team address");

if(!limitPaused)

{

if(!isWhiteListed[msg.sender])

{

require(value <= transferLimit, "transaction limit exceeded");

}

}

uint256 tokensToBurn = findFivePercent(value);

uint256 checkToBurn = totalBurn.add(tokensToBurn);

if(toBurn < checkToBurn)

{

tokensToBurn = toBurn.sub(totalBurn);

}

if(msg.sender==teamAddress)

{

uint256 currenttime = block.timestamp;

uint months = uint(((currenttime - releaseStartTime) / 60 / 60 / 24)).div(30);

uint256 releaseLimit = \_releasePerMonth.mul(months);

uint256 maxRelease = totalRelease.add(value);

require(releaseLimit >= maxRelease, "insufficient release balance");

if(tokensToBurn > 0)

{

require(tokensToBurn <= \_balances[burnAddress], "burn amount exceeds balance");

\_balances[burnAddress] = \_balances[burnAddress].sub(tokensToBurn);

\_totalSupply = \_totalSupply.sub(tokensToBurn);

totalBurn = totalBurn.add(tokensToBurn);

emit Transfer(burnAddress, address(0), tokensToBurn);

}

totalRelease=totalRelease.add(value);

\_balances[msg.sender] = \_balances[msg.sender].sub(value);

\_balances[to] = \_balances[to].add(value);

emit Transfer(msg.sender, to, value);

}

else

{

if(tokensToBurn > 0)

{

require(tokensToBurn <= \_balances[burnAddress], "burn amount exceeds balance");

\_balances[burnAddress] = \_balances[burnAddress].sub(tokensToBurn);

\_totalSupply = \_totalSupply.sub(tokensToBurn);

totalBurn = totalBurn.add(tokensToBurn);

emit Transfer(burnAddress, address(0), tokensToBurn);

}

\_balances[msg.sender] = \_balances[msg.sender].sub(value);

\_balances[to] = \_balances[to].add(value);

emit Transfer(msg.sender, to, value);

}

return true;

}

function airdrop(address[] memory receivers, uint256 amount) public {

require(msg.sender == airdropAddress, "airdrop address not found");

for (uint256 i = 0; i < receivers.length; i++) {

transfer(receivers[i], amount);

}

}

function approve(address spender, uint256 value) public returns (bool) {

require(spender != address(0));

\_allowed[msg.sender][spender] = value;

emit Approval(msg.sender, spender, value);

return true;

}

function transferFrom(address from, address to, uint256 value) public returns (bool) {

require(value <= \_balances[from], "transfer amount exceeds balance");

require(to != address(0), "can't transfer to the zero address");

require(to != burnAddress, "can't transfer to the burn address");

require(from != burnAddress, "can't transfer from the burn address");

require(to != teamAddress, "can't transfer to the team address");

require(value <= \_allowed[from][msg.sender], "allowed limit exceed");

if(!limitPaused)

{

if(!isWhiteListed[from])

{

require(value <= transferLimit, "transaction limit exceeded");

}

}

uint256 tokensToBurn = findFivePercent(value);

uint256 checkToBurn = totalBurn.add(tokensToBurn);

if(toBurn < checkToBurn)

{

tokensToBurn = toBurn.sub(totalBurn);

}

if(from==teamAddress)

{

uint256 currenttime = block.timestamp;

uint months = uint(((currenttime - releaseStartTime) / 60 / 60 / 24)).div(30);

uint256 releaseLimit = \_releasePerMonth.mul(months);

uint256 maxRelease = totalRelease.add(value);

require(releaseLimit >= maxRelease, "insufficient release balance");

if(tokensToBurn > 0)

{

require(tokensToBurn <= \_balances[burnAddress], "burn amount exceeds balance");

\_balances[burnAddress] = \_balances[burnAddress].sub(tokensToBurn);

\_totalSupply = \_totalSupply.sub(tokensToBurn);

totalBurn = totalBurn.add(tokensToBurn);

emit Transfer(burnAddress, address(0), tokensToBurn);

}

totalRelease=totalRelease.add(value);

\_allowed[from][msg.sender] = \_allowed[from][msg.sender].sub(value);

\_balances[from] = \_balances[from].sub(value);

\_balances[to] = \_balances[to].add(value);

emit Transfer(from, to, value);

}

else

{

if(tokensToBurn > 0)

{

require(tokensToBurn <= \_balances[burnAddress], "burn amount exceeds balance");

\_balances[burnAddress] = \_balances[burnAddress].sub(tokensToBurn);

\_totalSupply = \_totalSupply.sub(tokensToBurn);

totalBurn = totalBurn.add(tokensToBurn);

emit Transfer(burnAddress, address(0), tokensToBurn);

}

\_allowed[from][msg.sender] = \_allowed[from][msg.sender].sub(value);

\_balances[from] = \_balances[from].sub(value);

\_balances[to] = \_balances[to].add(value);

emit Transfer(from, to, value);

}

return true;

}

function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {

require(spender != address(0));

\_allowed[msg.sender][spender] = (\_allowed[msg.sender][spender].add(addedValue));

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

return true;

}

function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {

require(spender != address(0));

\_allowed[msg.sender][spender] = (\_allowed[msg.sender][spender].sub(subtractedValue));

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

return true;

}

function \_mint(address account, uint256 amount) internal {

require(amount != 0);

\_balances[account] = \_balances[account].add(amount);

emit Transfer(address(0), account, amount);

}

function monthlyTokenBurn() public returns (bool) {

require(msg.sender == burnAddress, "burn address not found");

uint256 currenttime = block.timestamp;

uint months = uint(((currenttime - contractTime) / 60 / 60 / 24)).div(30);

uint256 burnLimit = \_burnPerMonth.mul(months);

if(burnLimit > maxBurnLimit)

{

burnLimit = maxBurnLimit;

}

uint256 toNextBurn = burnLimit.sub(monthlyBurn);

require(toNextBurn != 0, "burn limit 0");

require(\_balances[burnAddress] != 0, "address balance 0");

monthlyBurn = burnLimit;

nextBurnTime = contractTime+((months+1)\*30 days);

\_balances[burnAddress] = \_balances[burnAddress].sub(toNextBurn);

\_totalSupply = \_totalSupply.sub(toNextBurn);

emit Transfer(burnAddress, address(0), toNextBurn);

return true;

}

}