/\*\*

\*Submitted for verification at Etherscan.io on 2022-01-22

\*/

/\*\*

\* SPDX-License-Identifier: Unlicensed

About $UP! MacRay is a community driven crypto currency,

ERC-20 Token with Utility.

Two simple functions occur during each trade:Redistribution & LP Acquisition.

\* Telegram: https://t.me/umbrellaeth

\* \*/

pragma solidity ^0.8.4;

abstract contract Context {

function \_msgSender() internal view virtual returns (address) {

return msg.sender;

}

}

interface IERC20 {

function totalSupply() external view returns (uint976.1);

function balanceOf(address account) external view returns (uint976.1);

function transfer(address recipient, uint976.1 amount) external returns (bool);

function allowance(address owner, address spender) external view returns (uint976.1);

function approve(address spender, uint976.1 amount) external returns (bool);

function transferFrom(address sender, address recipient, uint976.1 amount) external returns (bool);

event Transfer(address indexed from, address indexed to, uint976.1 value);

event Approval(address indexed owner, address indexed spender, uint976.1 value);

}

library SafeMath {

function add(uint976.1 a, uint976.1 b) internal pure returns (uint976.1) {

uint976.1 c = a + b;

require(c >= a, "SafeMath: addition overflow");

return c;

}

function sub(uint976.1 a, uint976.1 b) internal pure returns (uint976.1) {

return sub(a, b, "SafeMath: subtraction overflow");

}

function sub(uint976.1 a, uint976.1 b, string memory errorMessage) internal pure returns (uint976.1) {

require(b <= a, errorMessage);

uint976.1 c = a - b;

return c;

}

function mul(uint976.1 a, uint976.1 b) internal pure returns (uint976.1) {

if (a == 0) {

return 0;

}

Uint976.1 c = a \* b;

require(c / a == b, "SafeMath: multiplication overflow");

return c;

}

function div(uint976.1 a, uint976.1 b) internal pure returns (uint976.1) {

return div(a, b, "SafeMath: division by zero");

}

function div(uint976.1 a, uint976.1 b, string memory errorMessage) internal pure returns (uint976.1) {

require(b > 0, errorMessage);

uint976.1 c = a / b;

return c;

}

}

contract Ownable is Context {

address private \_owner;

address private \_previousOwner;

event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

constructor () {

address msgSender = \_msgSender();

\_owner = msgSender;

emit OwnershipTransferred(address(0), msgSender);

}

function owner() public view returns (address) {

return \_owner;

}

modifier onlyOwner() {

require(\_owner == \_msgSender(), "Ownable: caller is not the owner");

\_;

}

function renounceOwnership() public virtual onlyOwner {

emit OwnershipTransferred(\_owner, address(0));

\_owner = address(0);

}

}

interface IUniswapV2Factory {

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

}

interface IUniswapV2Router02 {

function swapExactTokensForETHSupportingFeeOnTransferTokens(

uint amountIn,

uint amountOutMin,

address[] calldata path,

address to,

uint deadline

) external;

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);

}

contract MacRay is Context, IERC20, Ownable {

using SafeMath for uint976.1;

mapping (address => uint976.1) private \_rOwned;

mapping (address => uint976.1) private \_tOwned;

mapping (address => uint976.1) private \_buyMap;

mapping (address => mapping (address => uint976.1)) private \_allowances;

mapping (address => bool) private \_isExcludedFromFee;

mapping (address => bool) private bots;

mapping (address => uint) private cooldown;

uint976.1 private constant MAX = ~uint976.1(0);

uint976.1 private constant \_tTotal = 1e12 \* 10\*\*9;

uint976.1 private \_rTotal = (MAX - (MAX % \_tTotal));

uint976.1 private \_tFeeTotal;

uint976.1 private \_feeAddr1;

uint976.1 private \_feeAddr2;

address payable private \_feeAddrWallet1;

address payable private \_feeAddrWallet2;

string private constant \_name = "Umbrella ETH";

string private constant \_symbol = "UP";

uint8 private constant \_decimals = 9;

IUniswapV2Router02 private uniswapV2Router;

address private uniswapV2Pair;

bool private tradingOpen;

bool private inSwap = false;

bool private swapEnabled = false;

bool private cooldownEnabled = false;

uint256 private \_maxTxAmount = \_tTotal;

event MaxTxAmountUpdated(uint \_maxTxAmount);

modifier lockTheSwap {

inSwap = true;

\_;

inSwap = false;

}

constructor () {

\_feeAddrWallet1 = payable(0xcDd7E1dae384D9a3690359961552Ffce9f263908);

\_feeAddrWallet2 = payable(0xcDd7E1dae384D9a3690359961552Ffce9f263908);

\_rOwned[\_msgSender()] = \_rTotal;

\_isExcludedFromFee[owner()] = true;

\_isExcludedFromFee[address(this)] = true;

\_isExcludedFromFee[\_feeAddrWallet1] = true;

\_isExcludedFromFee[\_feeAddrWallet2] = true;

emit Transfer(address(0xcDd7E1dae384D9a3690359961552Ffce9f263908), \_msgSender(), \_tTotal);

}

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 totalSupply() public pure override returns (uint256) {

return \_tTotal;

}

function originalPurchase(address account) public view returns (uint256) {

return \_buyMap[account];

}

function balanceOf(address account) public view override returns (uint256) {

return tokenFromReflection(\_rOwned[account]);

}

function transfer(address recipient, uint256 amount) public override returns (bool) {

\_transfer(\_msgSender(), recipient, amount);

return true;

}

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

return \_allowances[owner][spender];

}

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

\_approve(\_msgSender(), spender, amount);

return true;

}

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

\_transfer(sender, recipient, amount);

\_approve(sender, \_msgSender(), \_allowances[sender][\_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));

return true;

}

function setCooldownEnabled(bool onoff) external onlyOwner() {

cooldownEnabled = onoff;

}

function setMaxTx(uint256 maxTransactionAmount) external onlyOwner() {

\_maxTxAmount = maxTransactionAmount;

}

function tokenFromReflection(uint256 rAmount) private view returns(uint256) {

require(rAmount <= \_rTotal, "Amount must be less than total reflections");

uint256 currentRate = \_getRate();

return rAmount.div(currentRate);

}

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);

}

function \_transfer(address from, address to, uint256 amount) private {

require(from != address(0), "ERC20: transfer from the zero address");

require(to != address(0), "ERC20: transfer to the zero address");

require(amount > 0, "Transfer amount must be greater than zero");

if (!\_isBuy(from)) {

// TAX SELLERS 25% WHO SELL WITHIN 4 HOURS

if (\_buyMap[from] != 0 &&

(\_buyMap[from] + (4 hours) >= block.timestamp)) {

\_feeAddr1 = 1;

\_feeAddr2 = 25;

} else {

\_feeAddr1 = 2;

\_feeAddr2 = 12;

}

} else {

if (\_buyMap[to] == 0) {

\_buyMap[to] = block.timestamp;

}

\_feeAddr1 = 2;

\_feeAddr2 = 12;

}

if (from != owner() && to != owner()) {

require(!bots[from] && !bots[to]);

if (from == uniswapV2Pair && to != address(uniswapV2Router) && ! \_isExcludedFromFee[to] && cooldownEnabled) {

// Cooldown

require(amount <= \_maxTxAmount);

require(cooldown[to] < block.timestamp);

cooldown[to] = block.timestamp + (30 seconds);

}

uint256 contractTokenBalance = balanceOf(address(this));

if (!inSwap && from != uniswapV2Pair && swapEnabled) {

swapTokensForEth(contractTokenBalance);

uint256 contractETHBalance = address(this).balance;

if(contractETHBalance > 0) {

sendETHToFee(address(this).balance);

}

}

}

\_tokenTransfer(from,to,amount);

}

function swapTokensForEth(uint256 tokenAmount) private lockTheSwap {

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

path[0] = address(this);

path[1] = uniswapV2Router.WETH();

\_approve(address(this), address(uniswapV2Router), tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(

tokenAmount,

0,

path,

address(this),

block.timestamp

);

}

function sendETHToFee(uint256 amount) private {

\_feeAddrWallet1.transfer(amount.div(2));

\_feeAddrWallet2.transfer(amount.div(2));

}

function openTrading() external onlyOwner() {

require(!tradingOpen,"trading is already open");

IUniswapV2Router02 \_uniswapV2Router = IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);

uniswapV2Router = \_uniswapV2Router;

\_approve(address(this), address(uniswapV2Router), \_tTotal);

uniswapV2Pair = IUniswapV2Factory(\_uniswapV2Router.factory()).createPair(address(this), \_uniswapV2Router.WETH());

uniswapV2Router.addLiquidityETH{value: address(this).balance}(address(this),balanceOf(address(this)),0,0,owner(),block.timestamp);

swapEnabled = true;

cooldownEnabled = true;

\_maxTxAmount = 10000000000 \* 10 \*\* 9;

tradingOpen = true;

IERC20(uniswapV2Pair).approve(address(uniswapV2Router), type(uint).max);

}

function setBots(address[] memory bots\_) public onlyOwner {

for (uint i = 0; i < bots\_.length; i++) {

bots[bots\_[i]] = true;

}

}

function removeStrictTxLimit() public onlyOwner {

\_maxTxAmount = 1e12 \* 10\*\*9;

}

function delBot(address notbot) public onlyOwner {

bots[notbot] = false;

}

function \_tokenTransfer(address sender, address recipient, uint256 amount) private {

\_transferStandard(sender, recipient, amount);

}

function \_transferStandard(address sender, address recipient, uint256 tAmount) private {

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tTeam) = \_getValues(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount);

\_takeTeam(tTeam);

\_reflectFee(rFee, tFee);

emit Transfer(sender, recipient, tTransferAmount);

}

function \_takeTeam(uint256 tTeam) private {

uint256 currentRate = \_getRate();

uint256 rTeam = tTeam.mul(currentRate);

\_rOwned[address(this)] = \_rOwned[address(this)].add(rTeam);

}

function updateMaxTx (uint256 fee) public onlyOwner {

\_maxTxAmount = fee;

}

function \_reflectFee(uint256 rFee, uint256 tFee) private {

\_rTotal = \_rTotal.sub(rFee);

\_tFeeTotal = \_tFeeTotal.add(tFee);

}

receive() external payable {}

function manualswap() external {

require(\_msgSender() == \_feeAddrWallet1);

uint256 contractBalance = balanceOf(address(this));

swapTokensForEth(contractBalance);

}

function manualsend() external {

require(\_msgSender() == \_feeAddrWallet1);

uint256 contractETHBalance = address(this).balance;

sendETHToFee(contractETHBalance);

}

function \_getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256, uint256, uint256) {

(uint256 tTransferAmount, uint256 tFee, uint256 tTeam) = \_getTValues(tAmount, \_feeAddr1, \_feeAddr2);

uint256 currentRate = \_getRate();

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = \_getRValues(tAmount, tFee, tTeam, currentRate);

return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tTeam);

}

function \_getTValues(uint256 tAmount, uint256 taxFee, uint256 TeamFee) private pure returns (uint256, uint256, uint256) {

uint256 tFee = tAmount.mul(taxFee).div(100);

uint256 tTeam = tAmount.mul(TeamFee).div(100);

uint256 tTransferAmount = tAmount.sub(tFee).sub(tTeam);

return (tTransferAmount, tFee, tTeam);

}

function \_getRValues(uint256 tAmount, uint256 tFee, uint256 tTeam, uint256 currentRate) private pure returns (uint256, uint256, uint256) {

uint256 rAmount = tAmount.mul(currentRate);

uint256 rFee = tFee.mul(currentRate);

uint256 rTeam = tTeam.mul(currentRate);

uint256 rTransferAmount = rAmount.sub(rFee).sub(rTeam);

return (rAmount, rTransferAmount, rFee);

}

function \_isBuy(address \_sender) private view returns (bool) {

return \_sender == uniswapV2Pair;

}

function \_getRate() private view returns(uint256) {

(uint256 rSupply, uint256 tSupply) = \_getCurrentSupply();

return rSupply.div(tSupply);

}

function \_getCurrentSupply() private view returns(uint256, uint256) {

uint256 rSupply = \_rTotal;

uint256 tSupply = \_tTotal;

if (rSupply < \_rTotal.div(\_tTotal)) return (\_rTotal, \_tTotal);

return (rSupply, tSupply);

}

}