```
pragma solidity ^0.6.12;
// SPDX-License-Identifier: Unlicensed
interface IERC20 {

function totalSupply() external view returns (uint256);

/**
* @dev Returns the amount of tokens owned by `account`.
*/
function balanceOf(address account) external view returns (uint256);

/**
* @dev Moves `amount` tokens from the caller's account to `recipient`.
*
* Returns a boolean value indicating whether the operation succeeded.
*
* Emits a {Transfer} event.
*/
function transfer(address recipient, uint256 amount) external returns (bool);

/**
* @dev Returns the remaining number of tokens that `spender` will be
* allowed to spend on behalf of `owner` through {transferFrom}. This is
* zero by default.
*
* This value changes when {approve} or {transferFrom} are called.
*/
function allowance(address owner, address spender) external view returns (uint256);

/**
* @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
*
* Returns a boolean value indicating whether the operation succeeded.
*
* IMPORTANT: 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
*
* Emits an {Approval} event.
*/
function approve(address spender, uint256 amount) external returns (bool);

/**
* @dev Moves `amount` tokens from `sender` to `recipient` using the
* allowance mechanism. `amount` is then deducted from the caller's
* allowance.
*
* Returns a boolean value indicating whether the operation succeeded.
*
* Emits a {Transfer} event.
*/
```

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

/**
* @dev Emitted when `value` tokens are moved from one account (`from`) to
* another (`to`).
*
* Note that `value` may be zero.
*/
event Transfer(address indexed from, address indexed to, uint256 value);

/**
* @dev Emitted when the allowance of a `spender` for an `owner` is set by
* a call to {approve}. `value` is the new allowance.
*/
event Approval(address indexed owner, address indexed spender, uint256 value);
}



/**
* @dev Wrappers over Solidity's arithmetic operations with added overflow
* checks.
*
* Arithmetic operations in Solidity wrap on overflow. This can easily result
* in bugs, because programmers usually assume that an overflow raises an
* error, which is the standard behavior in high level programming languages.
* `SafeMath` restores this intuition by reverting the transaction when an
* operation overflows.
*
* Using this library instead of the unchecked operations eliminates an entire
* class of bugs, so it's recommended to use it always.
*/

library SafeMath {
/**
* @dev Returns the addition of two unsigned integers, reverting on
* overflow.
*
* Counterpart to Solidity's `+` operator.
*
* Requirements:
*
* - Addition cannot overflow.
*/
function add(uint256 a, uint256 b) internal pure returns (uint256) {
uint256 c = a + b;
require(c >= a, "SafeMath: addition overflow");

return c;
}

/**
* @dev Returns the subtraction of two unsigned integers, reverting on
* overflow (when the result is negative).
*
* Counterpart to Solidity's `-` operator.
*
* Requirements:
*
* - Subtraction cannot overflow.
*/
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
return sub(a, b, "SafeMath: subtraction overflow");
}

/**
* @dev Returns the subtraction of two unsigned integers, reverting with custom message on
* overflow (when the result is negative).
*
* Counterpart to Solidity's `-` operator.
*
* Requirements:
*
* - Subtraction cannot overflow.
*/
function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
require(b <= a, errorMessage);
uint256 c = a - b;

return c;
}

/**
* @dev Returns the multiplication of two unsigned integers, reverting on
* overflow.
*
* Counterpart to Solidity's `*` operator.
*
* Requirements:
*
* - Multiplication cannot overflow.
*/
function mul(uint256 a, uint256 b) internal pure returns (uint256) {
// Gas optimization: this is cheaper than requiring 'a' not being zero, but the
// benefit is lost if 'b' is also tested.
// See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
if (a == 0) {
return 0;
}

uint256 c = a * b;
require(c / a == b, "SafeMath: multiplication overflow");

return c;
}

/**
```

```
* @dev Returns the integer division of two unsigned integers. Reverts on
* division by zero. The result is rounded towards zero.
*
* Counterpart to Solidity's `/` operator. Note: this function uses a
* `revert` opcode (which leaves remaining gas untouched) while Solidity
* uses an invalid opcode to revert (consuming all remaining gas).
*
* Requirements:
*
* - The divisor cannot be zero.
*/
function div(uint256 a, uint256 b) internal pure returns (uint256) {
return div(a, b, "SafeMath: division by zero");
}

/**
* @dev Returns the integer division of two unsigned integers. Reverts with custom message on
* division by zero. The result is rounded towards zero.
*
* Counterpart to Solidity's `/` operator. Note: this function uses a
* `revert` opcode (which leaves remaining gas untouched) while Solidity
* uses an invalid opcode to revert (consuming all remaining gas).
*
* Requirements:
*
* - The divisor cannot be zero.
*/
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
require(b > 0, errorMessage);
uint256 c = a / b;
// assert(a == b * c + a % b); // There is no case in which this doesn't hold

return c;
}

/**
* @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
* Reverts when dividing by zero.
*
* Counterpart to Solidity's `%` operator. This function uses a `revert`
* opcode (which leaves remaining gas untouched) while Solidity uses an
* invalid opcode to revert (consuming all remaining gas).
*
* Requirements:
*
* - The divisor cannot be zero.
*/
function mod(uint256 a, uint256 b) internal pure returns (uint256) {
return mod(a, b, "SafeMath: modulo by zero");
}

/**
* @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
* Reverts with custom message when dividing by zero.
```

```
*
* Counterpart to Solidity's `%` operator. This function uses a `revert`
* opcode (which leaves remaining gas untouched) while Solidity uses an
* invalid opcode to revert (consuming all remaining gas).
*
* Requirements:
*
* - The divisor cannot be zero.
*/
function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
require(b != 0, errorMessage);
return a % b;
}
}

abstract contract Context {
function _msgSender() internal view virtual returns (address payable) {
return msg.sender;
}

function _msgData() internal view virtual returns (bytes memory) {
this; // silence state mutability warning without generating bytecode - see https://github.com/
ethereum/solidity/issues/2691
return msg.data;
}
}


/**
* @dev Collection of functions related to the address type
*/
library Address {
/**
* @dev Returns true if `account` is a contract.
*
* [IMPORTANT]
* ====
* It is unsafe to assume that an address for which this function returns
* false is an externally-owned account (EOA) and not a contract.
*
* Among others, `isContract` will return false for the following
* types of addresses:
*
*  - an externally-owned account
*  - a contract in construction
*  - an address where a contract will be created
*  - an address where a contract lived, but was destroyed
* ====
*/
function isContract(address account) internal view returns (bool) {
// According to EIP-1052, 0x0 is the value returned for not-yet created accounts
// and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
returned
// for accounts without code, i.e. `keccak256(")`
```

```
bytes32 codehash;
bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
// solhint-disable-next-line no-inline-assembly
assembly { codehash := extcodehash(account) }
return (codehash != accountHash && codehash != 0x0);
}

/**
* @dev Replacement for Solidity's `transfer`: sends `amount` wei to
* `recipient`, forwarding all available gas and reverting on errors.
*
* https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
* of certain opcodes, possibly making contracts go over the 2300 gas limit
* imposed by `transfer`, making them unable to receive funds via
* `transfer`. {sendValue} removes this limitation.
*
* https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
*
* IMPORTANT: because control is transferred to `recipient`, care must be
* taken to not create reentrancy vulnerabilities. Consider using
* {ReentrancyGuard} or the
* https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-
effects-interactions-pattern[checks-effects-interactions pattern].
*/
function sendValue(address payable recipient, uint256 amount) internal {
require(address(this).balance >= amount, "Address: insufficient balance");

// solhint-disable-next-line avoid-low-level-calls, avoid-call-value
(bool success, ) = recipient.call{ value: amount }("");
require(success, "Address: unable to send value, recipient may have reverted");
}

/**
* @dev Performs a Solidity function call using a low level `call`. A
* plain`call` is an unsafe replacement for a function call: use this
* function instead.
*
* If `target` reverts with a revert reason, it is bubbled up by this
* function (like regular Solidity function calls).
*
* Returns the raw returned data. To convert to the expected return value,
* use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?
highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
*
* Requirements:
*
* - `target` must be a contract.
* - calling `target` with `data` must not revert.
*
* _Available since v3.1._
*/
function functionCall(address target, bytes memory data) internal returns (bytes memory) {
return functionCall(target, data, "Address: low-level call failed");
```

```
}

/**
* @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
* `errorMessage` as a fallback revert reason when `target` reverts.
*
* _Available since v3.1._
*/
function functionCall(address target, bytes memory data, string memory errorMessage) internal
returns (bytes memory) {
return _functionCallWithValue(target, data, 0, errorMessage);
}

/**
* @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
* but also transferring `value` wei to `target`.
*
* Requirements:
*
* - the calling contract must have an ETH balance of at least `value`.
* - the called Solidity function must be `payable`.
*
* _Available since v3.1._
*/
function functionCallWithValue(address target, bytes memory data, uint256 value) internal
returns (bytes memory) {
return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
}

/**
* @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}
[`functionCallWithValue`], but
* with `errorMessage` as a fallback revert reason when `target` reverts.
*
* _Available since v3.1._
*/
function functionCallWithValue(address target, bytes memory data, uint256 value, string memory
errorMessage) internal returns (bytes memory) {
require(address(this).balance >= value, "Address: insufficient balance for call");
return _functionCallWithValue(target, data, value, errorMessage);
}

function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string
memory errorMessage) private returns (bytes memory) {
require(isContract(target), "Address: call to non-contract");

// solhint-disable-next-line avoid-low-level-calls
(bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
if (success) {
return returndata;
} else {
// Look for revert reason and bubble it up if present
if (returndata.length > 0) {
// The easiest way to bubble the revert reason is using memory via assembly
// solhint-disable-next-line no-inline-assembly
assembly {
let returndata_size := mload(returndata)
revert(add(32, returndata), returndata_size)
}
} else {
revert(errorMessage);
}
}
}
}

/**
* @dev Contract module which provides a basic access control mechanism, where
* there is an account (an owner) that can be granted exclusive access to
* specific functions.
*
* By default, the owner account will be the one that deploys the contract. This
* can later be changed with {transferOwnership}.
*
* This module is used through inheritance. It will make available the modifier
* `onlyOwner`, which can be applied to your functions to restrict their use to
* the owner.
*/
contract Ownable is Context {
address private _owner;
address private _previousOwner;
uint256 private _lockTime;

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

/**
* @dev Initializes the contract setting the deployer as the initial owner.
*/
constructor () internal {
address msgSender = _msgSender();
_owner = msgSender;
emit OwnershipTransferred(address(0), msgSender);
}

/**
* @dev Returns the address of the current owner.
*/
function owner() public view returns (address) {
return _owner;
}

/**
* @dev Throws if called by any account other than the owner.
*/
modifier onlyOwner() {
require(_owner == _msgSender(), "Ownable: caller is not the owner");
_;
```

```
}

/**
* @dev Leaves the contract without owner. It will not be possible to call
* `onlyOwner` functions anymore. Can only be called by the current owner.
*
* NOTE: Renouncing ownership will leave the contract without an owner,
* thereby removing any functionality that is only available to the owner.
*/
function renounceOwnership() public virtual onlyOwner {
emit OwnershipTransferred(_owner, address(0));
_owner = address(0);
}

/**
* @dev Transfers ownership of the contract to a new account (`newOwner`).
* Can only be called by the current owner.
*/
function transferOwnership(address newOwner) public virtual onlyOwner {
require(newOwner != address(0), "Ownable: new owner is the zero address");
emit OwnershipTransferred(_owner, newOwner);
_owner = newOwner;
}

function geUnlockTime() public view returns (uint256) {
return _lockTime;
}

//Locks the contract for owner for the amount of time provided
function lock(uint256 time) public virtual onlyOwner {
_previousOwner = _owner;
_owner = address(0);
_lockTime = now + time;
emit OwnershipTransferred(_owner, address(0));
}

//Unlocks the contract for owner when _lockTime is exceeds
function unlock() public virtual {
require(_previousOwner == msg.sender, "You don't have permission to unlock");
require(now > _lockTime , "Contract is locked until 7 days");
emit OwnershipTransferred(_owner, _previousOwner);
_owner = _previousOwner;
}
}

// pragma solidity >=0.5.0;

interface IUniswapV2Factory {
event PairCreated(address indexed token0, address indexed token1, address pair, uint);

function feeTo() external view returns (address);
function feeToSetter() external view returns (address);

function getPair(address tokenA, address tokenB) external view returns (address pair);
function allPairs(uint) external view returns (address pair);
function allPairsLength() external view returns (uint);

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

function setFeeTo(address) external;
function setFeeToSetter(address) external;
}


// pragma solidity >=0.5.0;

interface IUniswapV2Pair {
event Approval(address indexed owner, address indexed spender, uint value);
event Transfer(address indexed from, address indexed to, uint value);

function name() external pure returns (string memory);
function symbol() external pure returns (string memory);
function decimals() external pure returns (uint8);
function totalSupply() external view returns (uint);
function balanceOf(address owner) external view returns (uint);
function allowance(address owner, address spender) external view returns (uint);

function approve(address spender, uint value) external returns (bool);
function transfer(address to, uint value) external returns (bool);
function transferFrom(address from, address to, uint value) external returns (bool);

function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function nonces(address owner) external view returns (uint);

function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r,
bytes32 s) external;

event Mint(address indexed sender, uint amount0, uint amount1);
event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
event Swap(
address indexed sender,
uint amount0In,
uint amount1In,
uint amount0Out,
uint amount1Out,
address indexed to
);
event Sync(uint112 reserve0, uint112 reserve1);

function MINIMUM_LIQUIDITY() external pure returns (uint);
function factory() external view returns (address);
function token0() external view returns (address);
function token1() external view returns (address);
function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32
blockTimestampLast);
function price0CumulativeLast() external view returns (uint);
function price1CumulativeLast() external view returns (uint);
function kLast() external view returns (uint);

function mint(address to) external returns (uint liquidity);
function burn(address to) external returns (uint amount0, uint amount1);
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
function skim(address to) external;
function sync() external;

function initialize(address, address) external;
}

// pragma solidity >=0.6.2;

interface IUniswapV2Router01 {
function factory() external pure returns (address);
function WETH() external pure returns (address);

function addLiquidity(
address tokenA,
address tokenB,
uint amountADesired,
uint amountBDesired,
uint amountAMin,
uint amountBMin,
address to,
uint deadline
) external returns (uint amountA, uint amountB, uint liquidity);
function addLiquidityETH(
address token,
uint amountTokenDesired,
uint amountTokenMin,
uint amountETHMin,
address to,
uint deadline
) external payable returns (uint amountToken, uint amountETH, uint liquidity);
function removeLiquidity(
address tokenA,
address tokenB,
uint liquidity,
uint amountAMin,
uint amountBMin,
address to,
uint deadline
) external returns (uint amountA, uint amountB);
function removeLiquidityETH(
address token,
uint liquidity,
uint amountTokenMin,
uint amountETHMin,
address to,
uint deadline
) external returns (uint amountToken, uint amountETH);
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 removeLiquidityETHWithPermit(
address token,
uint liquidity,
uint amountTokenMin,
uint amountETHMin,
address to,
uint deadline,
bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountToken, uint amountETH);
function swapExactTokensForTokens(
uint amountIn,
uint amountOutMin,
address[] calldata path,
address to,
uint deadline
) external returns (uint[] memory amounts);
function swapTokensForExactTokens(
uint amountOut,
uint amountInMax,
address[] calldata path,
address to,
uint deadline
) external returns (uint[] memory amounts);
function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
external
payable
returns (uint[] memory amounts);
function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
external
returns (uint[] memory amounts);
function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
external
returns (uint[] memory amounts);
function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
external
payable
returns (uint[] memory amounts);

function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint amountOut);
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint
amountIn);
function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory amounts);
function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory amounts);
}



// pragma solidity >=0.6.2;

interface IUniswapV2Router02 is IUniswapV2Router01 {
function removeLiquidityETHSupportingFeeOnTransferTokens(
address token,
uint liquidity,
uint amountTokenMin,
uint amountETHMin,
address to,
uint deadline
) external returns (uint amountETH);
function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
address token,
uint liquidity,
uint amountTokenMin,
uint amountETHMin,
address to,
uint deadline,
bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountETH);

function swapExactTokensForTokensSupportingFeeOnTransferTokens(
uint amountIn,
uint amountOutMin,
address[] calldata path,
address to,
uint deadline
) external;
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 SafeMoonBUNKIE is Context, IERC20, Ownable {
```

```
using SafeMath for uint256;
using Address for address;

mapping (address => uint256) private _rOwned;
mapping (address => uint256) private _tOwned;
mapping (address => mapping (address => uint256)) private _allowances;

mapping (address => bool) private _isExcludedFromFee;

mapping (address => bool) private _isExcluded;
address[] private _excluded;

uint256 private constant MAX = ~uint256(0);
uint256 private _tTotal = 1000000000 * 10**6 * 10**9;
uint256 private _rTotal = (MAX - (MAX % _tTotal));
uint256 private _tFeeTotal;

string private _name = "SafeMooBunkie-chan";
string private _symbol = "SAFEMOONBUNKIE";
uint8 private _decimals = 9;

uint256 public _taxFee = 5;
uint256 private _previousTaxFee = _taxFee;

uint256 public _liquidityFee = 5;
uint256 private _previousLiquidityFee = _liquidityFee;

IUniswapV2Router02 public immutable uniswapV2Router;
address public immutable uniswapV2Pair;

bool inSwapAndLiquify;
bool public swapAndLiquifyEnabled = true;

uint256 public _maxTxAmount = 5000000 * 10**6 * 10**9;
uint256 private numTokensSellToAddToLiquidity = 500000 * 10**6 * 10**9;

event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
event SwapAndLiquifyEnabledUpdated(bool enabled);
event SwapAndLiquify(
uint256 tokensSwapped,
uint256 ethReceived,
uint256 tokensIntoLiqudity
);

modifier lockTheSwap {
inSwapAndLiquify = true;
_;
inSwapAndLiquify = false;
}

constructor () public {
_rOwned[_msgSender()] = _rTotal;

IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(0x05fF2B0DB69458A0750badebc4f9e13aDd608C7FD99D1c33F9fC3444f
8101754aBC46c52416550D1);
// Create a uniswap pair for this new token
uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
.createPair(address(this), _uniswapV2Router.WETH());

// set the rest of the contract variables
uniswapV2Router = _uniswapV2Router;

//exclude owner and this contract from fee
_isExcludedFromFee[owner()] = true;
_isExcludedFromFee[address(this)] = true;

emit Transfer(address(0), _msgSender(), _tTotal);
}

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 override returns (uint256) {
return _tTotal;
}

function balanceOf(address account) public view override returns (uint256) {
if (_isExcluded[account]) return _tOwned[account];
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 increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
_approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
return true;
}

function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns
(bool) {
_approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue,
"ERC20: decreased allowance below zero"));
return true;
}

function isExcludedFromReward(address account) public view returns (bool) {
return _isExcluded[account];
}

function totalFees() public view returns (uint256) {
return _tFeeTotal;
}

function deliver(uint256 tAmount) public {
address sender = _msgSender();
require(!_isExcluded[sender], "Excluded addresses cannot call this function");
(uint256 rAmount,,,,,) = _getValues(tAmount);
_rOwned[sender] = _rOwned[sender].sub(rAmount);
_rTotal = _rTotal.sub(rAmount);
_tFeeTotal = _tFeeTotal.add(tAmount);
}

function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view
returns(uint256) {
require(tAmount <= _tTotal, "Amount must be less than supply");
if (!deductTransferFee) {
(uint256 rAmount,,,,,) = _getValues(tAmount);
return rAmount;
} else {
(,uint256 rTransferAmount,,,,) = _getValues(tAmount);
return rTransferAmount;
}
}

function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
require(rAmount <= _rTotal, "Amount must be less than total reflections");
uint256 currentRate =  _getRate();
return rAmount.div(currentRate);
}

function excludeFromReward(address account) public onlyOwner() {
// require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not exclude
Uniswap router.');
require(!_isExcluded[account], "Account is already excluded");
if(_rOwned[account] > 0) {
_tOwned[account] = tokenFromReflection(_rOwned[account]);
}
_isExcluded[account] = true;
_excluded.push(account);
}

function includeInReward(address account) external onlyOwner() {
require(_isExcluded[account], "Account is already excluded");
for (uint256 i = 0; i < _excluded.length; i++) {
if (_excluded[i] == account) {
_excluded[i] = _excluded[_excluded.length - 1];
_tOwned[account] = 0;
_isExcluded[account] = false;
_excluded.pop();
break;
}
}
}
function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256
tFee, uint256 tLiquidity) = _getValues(tAmount);
_tOwned[sender] = _tOwned[sender].sub(tAmount);
_rOwned[sender] = _rOwned[sender].sub(rAmount);
_tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
_rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
_takeLiquidity(tLiquidity);
_reflectFee(rFee, tFee);
emit Transfer(sender, recipient, tTransferAmount);
}

function excludeFromFee(address account) public onlyOwner {
_isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner {
_isExcludedFromFee[account] = false;
}

function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
_taxFee = taxFee;
}

function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
_liquidityFee = liquidityFee;
}

function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
_maxTxAmount = _tTotal.mul(maxTxPercent).div(
10**2
);
}
```

```
function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
swapAndLiquifyEnabled = _enabled;
emit SwapAndLiquifyEnabledUpdated(_enabled);
}

//to recieve ETH from uniswapV2Router when swaping
receive() external payable {}

function _reflectFee(uint256 rFee, uint256 tFee) private {
_rTotal = _rTotal.sub(rFee);
_tFeeTotal = _tFeeTotal.add(tFee);
}

function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256,
uint256, uint256) {
(uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getTValues(tAmount);
(uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(tAmount, tFee,
tLiquidity, _getRate());
return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tLiquidity);
}

function _getTValues(uint256 tAmount) private view returns (uint256, uint256, uint256) {
uint256 tFee = calculateTaxFee(tAmount);
uint256 tLiquidity = calculateLiquidityFee(tAmount);
uint256 tTransferAmount = tAmount.sub(tFee).sub(tLiquidity);
return (tTransferAmount, tFee, tLiquidity);
}

function _getRValues(uint256 tAmount, uint256 tFee, uint256 tLiquidity, uint256 currentRate)
private pure returns (uint256, uint256, uint256) {
uint256 rAmount = tAmount.mul(currentRate);
uint256 rFee = tFee.mul(currentRate);
uint256 rLiquidity = tLiquidity.mul(currentRate);
uint256 rTransferAmount = rAmount.sub(rFee).sub(rLiquidity);
return (rAmount, rTransferAmount, rFee);
}

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;
for (uint256 i = 0; i < _excluded.length; i++) {
if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
rSupply = rSupply.sub(_rOwned[_excluded[i]]);
tSupply = tSupply.sub(_tOwned[_excluded[i]]);
}
if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
return (rSupply, tSupply);
}
```

```
function _takeLiquidity(uint256 tLiquidity) private {
uint256 currentRate =  _getRate();
uint256 rLiquidity = tLiquidity.mul(currentRate);
_rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity);
if(_isExcluded[address(this)])
_tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity);
}

function calculateTaxFee(uint256 _amount) private view returns (uint256) {
return _amount.mul(_taxFee).div(
10**2
);
}

function calculateLiquidityFee(uint256 _amount) private view returns (uint256) {
return _amount.mul(_liquidityFee).div(
10**2
);
}

function removeAllFee() private {
if(_taxFee == 0 && _liquidityFee == 0) return;

_previousTaxFee = _taxFee;
_previousLiquidityFee = _liquidityFee;

_taxFee = 0;
_liquidityFee = 0;
}

function restoreAllFee() private {
_taxFee = _previousTaxFee;
_liquidityFee = _previousLiquidityFee;
}

function isExcludedFromFee(address account) public view returns(bool) {
return _isExcludedFromFee[account];
}

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(from != owner() && to != owner())
require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");

// is the token balance of this contract address over the min number of
// tokens that we need to initiate a swap + liquidity lock?
// also, don't get caught in a circular liquidity event.
// also, don't swap & liquify if sender is uniswap pair.
uint256 contractTokenBalance = balanceOf(address(this));

if(contractTokenBalance >= _maxTxAmount)
{
contractTokenBalance = _maxTxAmount;
}

bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
if (
overMinTokenBalance &&
!inSwapAndLiquify &&
from != uniswapV2Pair &&
swapAndLiquifyEnabled
) {
contractTokenBalance = numTokensSellToAddToLiquidity;
//add liquidity
swapAndLiquify(contractTokenBalance);
}

//indicates if fee should be deducted from transfer
bool takeFee = true;

//if any account belongs to _isExcludedFromFee account then remove the fee
if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
takeFee = false;
}

//transfer amount, it will take tax, burn, liquidity fee
_tokenTransfer(from,to,amount,takeFee);
}

function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
// split the contract balance into halves
uint256 half = contractTokenBalance.div(2);
uint256 otherHalf = contractTokenBalance.sub(half);

// capture the contract's current ETH balance.
// this is so that we can capture exactly the amount of ETH that the
// swap creates, and not make the liquidity event include any ETH that
// has been manually sent to the contract
uint256 initialBalance = address(this).balance;

// swap tokens for ETH
swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered
```

```
// how much ETH did we just swap into?
uint256 newBalance = address(this).balance.sub(initialBalance);

// add liquidity to uniswap
addLiquidity(otherHalf, newBalance);

emit SwapAndLiquify(half, newBalance, otherHalf);
}

function swapTokensForEth(uint256 tokenAmount) private {
// generate the uniswap pair path of token -> weth
address[] memory path = new address[](2);
path[0] = address(this);
path[1] = uniswapV2Router.WETH();

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

// make the swap
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
tokenAmount,
0, // accept any amount of ETH
path,
address(this),
block.timestamp
);
}

function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
// approve token transfer to cover all possible scenarios
_approve(address(this), address(uniswapV2Router), tokenAmount);

// add the liquidity
uniswapV2Router.addLiquidityETH{value: ethAmount}(
address(this),
tokenAmount,
0, // slippage is unavoidable
0, // slippage is unavoidable
owner(),
block.timestamp
);
}

//this method is responsible for taking all fee, if takeFee is true
function _tokenTransfer(address sender, address recipient, uint256 amount,bool takeFee) private
{
if(!takeFee)
removeAllFee();

if (_isExcluded[sender] && !_isExcluded[recipient]) {
_transferFromExcluded(sender, recipient, amount);
} else if (!_isExcluded[sender] && _isExcluded[recipient]) {
_transferToExcluded(sender, recipient, amount);
} else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
_transferStandard(sender, recipient, amount);
} else if (_isExcluded[sender] && _isExcluded[recipient]) {
_transferBothExcluded(sender, recipient, amount);
} else {
_transferStandard(sender, recipient, amount);
}

if(!takeFee)
restoreAllFee();
}

function _transferStandard(address sender, address recipient, uint256 tAmount) private {
(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256
tFee, uint256 tLiquidity) = _getValues(tAmount);
_rOwned[sender] = _rOwned[sender].sub(rAmount);
_rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
_takeLiquidity(tLiquidity);
_reflectFee(rFee, tFee);
emit Transfer(sender, recipient, tTransferAmount);
}

function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256
tFee, uint256 tLiquidity) = _getValues(tAmount);
_rOwned[sender] = _rOwned[sender].sub(rAmount);
_tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
_rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
_takeLiquidity(tLiquidity);
_reflectFee(rFee, tFee);
emit Transfer(sender, recipient, tTransferAmount);
}

function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256
tFee, uint256 tLiquidity) = _getValues(tAmount);
_tOwned[sender] = _tOwned[sender].sub(tAmount);
_rOwned[sender] = _rOwned[sender].sub(rAmount);
_rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
_takeLiquidity(tLiquidity);
_reflectFee(rFee, tFee);
emit Transfer(sender, recipient, tTransferAmount);
}




}
```

Removed: 4
Added: 4