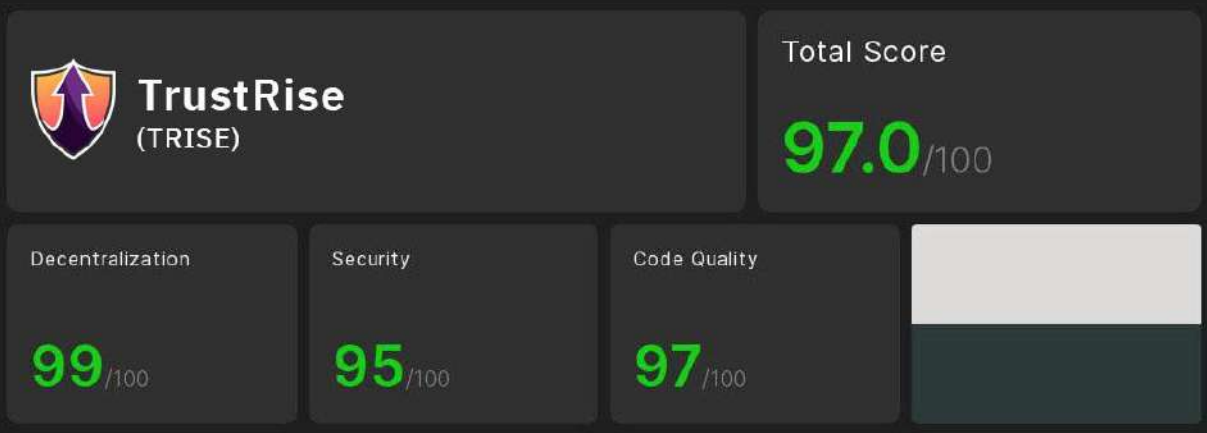

Warnings

Owner entitled to exclude from reward [564 - 564]

The owner can exclude an address from receiving rewards.

Owner entitled to include in reward [574 - 574]

The owner can include an address in receiving rewards.

```
pragma solidity ^0.8.0;


interface IBEP20 {

 function totalSupply() external view returns (uint256);

 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);
}


library SafeMath {

 function add(uint256 a, uint256 b) internal pure returns (uint256) {
 uint256 c = a + b;
 require(c >= a, "SafeMath: addition overflow");

 return c;
 }

 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
 return sub(a, b, "SafeMath: subtraction overflow");
 }

 function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
 require(b <= a, errorMessage);
 uint256 c = a - b;

 return c;
 }

 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
 if (a == 0) {
 return 0;
 }

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

 return c;
```

```
56 }
57
58 function div(uint256 a, uint256 b) internal pure returns (uint256) {
59 return div(a, b, "SafeMath: division by zero");
60 }
61
62 function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
63 require(b > 0, errorMessage);
64 uint256 c = a / b;
65
66 return c;
67 }
68
69 function mod(uint256 a, uint256 b) internal pure returns (uint256) {
70 return mod(a, b, "SafeMath: modulo by zero");
71 }
72
73 function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
74 require(b != 0, errorMessage);
75 return a % b;
76 }
77}
78
79abstract contract Context {
80 function _msgSender() internal view virtual returns (address) {
81 return msg.sender;
82 }
83
84 function _msgData() internal view virtual returns (bytes memory) {
85 this;
86 return msg.data;
87 }
88}
89
90
91library Address {
92
93 function isContract(address account) internal view returns (bool) {
94
95 bytes32 codehash;
96 bytes32 accountHash =;
97 assembly { codehash := extcodehash(account) }
98 return (codehash != accountHash && codehash != 0x0);
99 }
100
101 function sendValue(address payable recipient, uint256 amount) internal {
102 require(address(this).balance >= amount, "Address: insufficient balance");
103
104 (bool success, ) = recipient.call{ value: amount }("");
105 require(success, "Address: unable to send value, recipient may have reverted");
106 }
107
108 function functionCall(address target, bytes memory data) internal returns (bytes memory) {
109 return functionCall(target, data, "Address: low-level call failed");
110 }
111
112 function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
113 return _functionCallWithValue(target, data, 0, errorMessage);
114 }
115
116 function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
117 return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
118 }
119
120 function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns
(bytes memory) {
121 require(address(this).balance >= value, "Address: insufficient balance for call");
122 return _functionCallWithValue(target, data, value, errorMessage);
123 }
124
125 function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private retu
rns (bytes memory) {
126 require(isContract(target), "Address: call to non-contract");
127
128 (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
129 if (success) {
130 return returndata;
131 } else {
132 if (returndata.length > 0) {
133
134 assembly {
135 let returndata_size := mload(returndata)
136 revert(add(32, returndata), returndata_size)
137 }
138 } else {
139 revert(errorMessage);
140 }
141 }
142 }
143}
144
145contract Ownable is Context {
146 address private _owner;
147 address private _previousOwner;
148 uint256 private _lockTime;
```

```
149
150 event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
151
152 constructor () {
153 address msgSender = _msgSender();
154 _owner = msg.sender;
155 emit OwnershipTransferred(address(0), msgSender);
156 }
157
158 function owner() public view returns (address) {
159 return _owner;
160 }
161
162 modifier onlyOwner() {
163 require(_owner == _msgSender(), "Ownable: caller is not the owner");
164 _;
165 }
166
167 function renounceOwnership() public virtual onlyOwner {
168 emit OwnershipTransferred(_owner, address(0));
169 _owner = address(0);
170 }
171
172 function transferOwnership(address newOwner) public virtual onlyOwner {
173 require(newOwner != address(0), "Ownable: new owner is the zero address");
174 emit OwnershipTransferred(_owner, newOwner);
175 _owner = newOwner;
176 }
177
178}
179
180// pragma solidity >=0.5.0;
181
182interface IUniswapV2Factory {
183 event PairCreated(address indexed token0, address indexed token1, address pair, uint);
184
185 function feeTo() external view returns (address);
186 function feeToSetter() external view returns (address);
187
188 function getPair(address tokenA, address tokenB) external view returns (address pair);
189 function allPairs(uint) external view returns (address pair);
190 function allPairsLength() external view returns (uint);
191
192 function createPair(address tokenA, address tokenB) external returns (address pair);
193
194 function setFeeTo(address) external;
195 function setFeeToSetter(address) external;
196}
197
198
199// pragma solidity >=0.5.0;
200
201interface IUniswapV2Pair {
202 event Approval(address indexed owner, address indexed spender, uint value);
203 event Transfer(address indexed from, address indexed to, uint value);
204
205 function name() external pure returns (string memory);
206 function symbol() external pure returns (string memory);
207 function decimals() external pure returns (uint8);
208 function totalSupply() external view returns (uint);
209 function balanceOf(address owner) external view returns (uint);
210 function allowance(address owner, address spender) external view returns (uint);
211
212 function approve(address spender, uint value) external returns (bool);
213 function transfer(address to, uint value) external returns (bool);
214 function transferFrom(address from, address to, uint value) external returns (bool);
215
216 function DOMAIN_SEPARATOR() external view returns (bytes32);
217 function PERMIT_TYPEHASH() external pure returns (bytes32);
218 function nonces(address owner) external view returns (uint);
219
220 function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;
221
222 event Mint(address indexed sender, uint amount0, uint amount1);
223 event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
224 event Swap(
225 address indexed sender,
226 uint amount0In,
227 uint amount1In,
228 uint amount0Out,
229 uint amount1Out,
230 address indexed to
231 );
232 event Sync(uint112 reserve0, uint112 reserve1);
233
234 function MINIMUM_LIQUIDITY() external pure returns (uint);
235 function factory() external view returns (address);
236 function token0() external view returns (address);
237 function token1() external view returns (address);
238 function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
239 function price0CumulativeLast() external view returns (uint);
240 function price1CumulativeLast() external view returns (uint);
241 function kLast() external view returns (uint);
242
243 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
```

```
339 returns (uint[] memory amounts);
340
341 function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
342 function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint amountOut);
343 function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint amountIn);
344 function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory amounts);
345 function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory amounts);
346}
347
348
349
350interface IUniswapV2Router02 is IUniswapV2Router01 {
351 function removeLiquidityETHSupportingFeeOnTransferTokens(
352 address token,
353 uint liquidity,
354 uint amountTokenMin,
355 uint amountETHMin,
356 address to,
357 uint deadline
358 ) external returns (uint amountETH);
359 function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
360 address token,
361 uint liquidity,
362 uint amountTokenMin,
363 uint amountETHMin,
364 address to,
365 uint deadline,
366 bool approveMax, uint8 v, bytes32 r, bytes32 s
367 ) external returns (uint amountETH);
368
369 function swapExactTokensForTokensSupportingFeeOnTransferTokens(
370 uint amountIn,
371 uint amountOutMin,
372 address[] calldata path,
373 address to,
374 uint deadline
375 ) external;
376 function swapExactETHForTokensSupportingFeeOnTransferTokens(
377 uint amountOutMin,
378 address[] calldata path,
379 address to,
380 uint deadline
381 ) external payable;
382 function swapExactTokensForETHSupportingFeeOnTransferTokens(
383 uint amountIn,
384 uint amountOutMin,
385 address[] calldata path,
386 address to,
387 uint deadline
388 ) external;
389}
390
391
392contract TrustRise is Context, IBEP20, Ownable {
393 using SafeMath for uint256;
394 using Address for address;
395
396 mapping (address => uint256) private _rOwned;
397 mapping (address => uint256) private _tOwned;
398 mapping (address => mapping (address => uint256)) private _allowances;
399 mapping (address => bool) AutomaticMarketMaker;
400 mapping (address => bool) private _isExcludedFromFee;
401 address public CharityWallet =;
402 mapping (address => bool) private _isExcluded;
403 address[] private _excluded;
404
405 uint256 private constant MAX = ~uint256(0);
406 uint256 private _tTotal = 125 *10**8 * 10**18;
407 uint256 private _rTotal = (MAX - (MAX % _tTotal));
408 uint256 private _tFeeTotal;
409
410 string private _name = "TrustRise";
411 string private _symbol = "TRISE";
412 uint8 private _decimals = 18;
413
414 uint256 private _taxFee = 2;
415 uint256 private _previousTaxFee = _taxFee;
416
417 uint256 private _liquidityFee = 2;
418 uint256 private _previousLiquidityFee = _liquidityFee;
419
420 uint256 private _CharityFee = 2;
421 uint256 private _previousCharityFee = _CharityFee;
422
423
424 uint256 public buyTaxFee = _taxFee;
425 uint256 public buyLiquidityFee = _liquidityFee;
426 uint256 public buyCharityFee = _CharityFee;
427
428
429 uint256 public SellTaxFee = 2;
430 uint256 public SellLiquidityFee = 2;
431 uint256 public SellCharityFee = 2;
432
433
```

```
434
435 IUniswapV2Router02 public immutable uniswapV2Router;
436 address public immutable uniswapV2Pair;
437
438 bool inSwapAndLiquify;
439 bool public swapAndLiquifyEnabled = true;
440
441 uint256 public _maxTxAmount = 125 *10**8 * 10**18;
442 uint256 private numTokensSellToAddToLiquidity = 25 *10**8 * 10**18;
443
444 event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
445 event SwapAndLiquifyEnabledUpdated(bool enabled);
446 event SwapAndLiquify(
447 uint256 tokensSwapped,
448 uint256 ethReceived,
449 uint256 tokensIntoLiqudity
450 );
451
452 modifier lockTheSwap {
453 inSwapAndLiquify = true;
454 _;
455 inSwapAndLiquify = false;
456 }
457
458 constructor () {
459 _rOwned[_msgSender()] = _rTotal;
460 IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02();
461
462 uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
463 .createPair(address(this), _uniswapV2Router.WETH());
464
465 uniswapV2Router = _uniswapV2Router;
466
467 _isExcludedFromFee[owner()] = true;
468 _isExcludedFromFee[address(this)] = true;
469 _isExcludedFromFee[CharityWallet] = true;
470 AutomaticMarketMaker[uniswapV2Pair] = true;
471 emit Transfer(address(0), _msgSender(), _tTotal);
472 }
473
474 function name() public view returns (string memory) {
475 return _name;
476 }
477
478 function symbol() public view returns (string memory) {
479 return _symbol;
480 }
481
482 function decimals() public view returns (uint8) {
483 return _decimals;
484 }
485
486 function totalSupply() public view override returns (uint256) {
487 return _tTotal;
488 }
489
490 function balanceOf(address account) public view override returns (uint256) {
491 if (_isExcluded[account]) return _tOwned[account];
492 return tokenFromReflection(_rOwned[account]);
493 }
494
495 function transfer(address recipient, uint256 amount) public override returns (bool) {
496 _transfer(_msgSender(), recipient, amount);
497 return true;
498 }
499
500 function allowance(address owner, address spender) public view override returns (uint256) {
501 return _allowances[owner][spender];
502 }
503
504 function approve(address spender, uint256 amount) public override returns (bool) {
505 _approve(_msgSender(), spender, amount);
506 return true;
507 }
508
509 function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
510 _transfer(sender, recipient, amount);
511 _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "BEP20: transfer amount exceeds allowance"));
512 return true;
513 }
514
515 function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
516 _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
517 return true;
518 }
519
520 function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
521 _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "BEP20: decreased allowance below zer
o"));
522 return true;
523 }
524
525 function isExcludedFromReward(address account) public view returns (bool) {
526 return _isExcluded[account];
527 }
```

```
528
529 function totalFees() public view returns (uint256) {
530 return _tFeeTotal;
531 }
532 function setAutomaticMarketMaker(address _autoMM,bool _status) external onlyOwner(){
533 AutomaticMarketMaker[_autoMM] = _status;
534 }
535 function _isAutomaticMarketMaker(address AMM)public view returns(bool){
536 return AutomaticMarketMaker[AMM];
537 }
538 function deliver(uint256 tAmount) public {
539 address sender = _msgSender();
540 require(!_isExcluded[sender], "Excluded addresses cannot call this function");
541 (uint256 rAmount,,,,,) = _getValues(tAmount);
542 _rOwned[sender] = _rOwned[sender].sub(rAmount);
543 _rTotal = _rTotal.sub(rAmount);
544 _tFeeTotal = _tFeeTotal.add(tAmount);
545 }
546
547 function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns(uint256) {
548 require(tAmount <= _tTotal, "Amount must be less than supply");
549 if (!deductTransferFee) {
550 (uint256 rAmount,,,,,) = _getValues(tAmount);
551 return rAmount;
552 } else {
553 (,uint256 rTransferAmount,,,,) = _getValues(tAmount);
554 return rTransferAmount;
555 }
556 }
557
558 function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
559 require(rAmount <= _rTotal, "Amount must be less than total reflections");
560 uint256 currentRate = _getRate();
561 return rAmount.div(currentRate);
562 }
563
564 function excludeFromReward(address account) public onlyOwner() {
565 require(account !=, 'We can not exclude Pancake router.');
566 require(!_isExcluded[account], "Account is already excluded");
567 if(_rOwned[account] > 0) {
568 _tOwned[account] = tokenFromReflection(_rOwned[account]);
569 }
570 _isExcluded[account] = true;
571 _excluded.push(account);
572 }
573
574 function includeInReward(address account) external onlyOwner() {
575 require(_isExcluded[account], "Account is already excluded");
576 for (uint256 i = 0; i < _excluded.length; i++) {
577 if (_excluded[i] == account) {
578 _excluded[i] = _excluded[_excluded.length - 1];
579 _tOwned[account] = 0;
580 _isExcluded[account] = false;
581 _excluded.pop();
582 break;
583 }
584 }
585 }
586 function excludeFromFee(address account) public onlyOwner {
587 _isExcludedFromFee[account] = true;
588 }
589
590 function includeInFee(address account) public onlyOwner {
591 _isExcludedFromFee[account] = false;
592 }
593
594 function setCharityWallet(address newWallet) external onlyOwner() {
595 CharityWallet = newWallet;
596 _isExcludedFromFee[CharityWallet] = true;
597 }
598 function setMaxTxAmount(uint256 maxTxAmount) external onlyOwner(){
599 _maxTxAmount = maxTxAmount * 10e18;
600 }
601
602 function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
603 swapAndLiquifyEnabled = _enabled;
604 emit SwapAndLiquifyEnabledUpdated(_enabled);
605 }
606 function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
607 (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getVal
ues(tAmount);
608 _tOwned[sender] = _tOwned[sender].sub(tAmount);
609 _rOwned[sender] = _rOwned[sender].sub(rAmount);
610 _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
611 _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
612 _takeLiquidity(tLiquidity);
613 _reflectFee(rFee, tFee);
614 emit Transfer(sender, recipient, tTransferAmount);
615 }
616
617
618
619 receive() external payable {}
620
621 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, u
int256, 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;
_previousCharityFee = _CharityFee;
_previousLiquidityFee = _liquidityFee;

_taxFee = 0;
_CharityFee = 0;
_liquidityFee = 0;

}

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

}
function setbuyTaxFees(uint256 taxFee,uint256 CharityFee,uint256 liquidityFee)external onlyOwner(){
buyTaxFee = taxFee;
buyCharityFee = CharityFee;
buyLiquidityFee = liquidityFee;

_taxFee = taxFee;
_liquidityFee = liquidityFee;
_CharityFee = CharityFee;

_previousLiquidityFee = _liquidityFee;
_previousCharityFee = _CharityFee;
_previousTaxFee = _taxFee;

```

```
}
function setSellTaxFees(uint256 taxFee,uint256 CharityFee,uint256 liquidityFee)external onlyOwner(){
SellTaxFee = taxFee;
SellCharityFee = CharityFee;
SellLiquidityFee = liquidityFee;

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

function _approve(address owner, address spender, uint256 amount) private {
require(owner != address(0), "BEP20: approve from the zero address");
require(spender != address(0), "BEP20: 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), "BEP20: transfer from the zero address");
require(to != address(0), "BEP20: transfer to the zero address");
require(amount > 0, "Transfer amount must be greater than zero");

uint256 contractTokenBalance = balanceOf(address(this));
bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
if (
overMinTokenBalance &&
!inSwapAndLiquify &&
from != uniswapV2Pair &&
swapAndLiquifyEnabled
) {
contractTokenBalance = numTokensSellToAddToLiquidity;
swapAndLiquify(contractTokenBalance);
}

_tokenTransfer(from,to,amount);
}

function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
uint256 half = contractTokenBalance.div(2);
uint256 otherHalf = contractTokenBalance.sub(half);

uint256 initialBalance = address(this).balance;

swapTokensForEth(half);

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

addLiquidity(otherHalf, newBalance);

emit SwapAndLiquify(half, newBalance, otherHalf);
}

function swapTokensForEth(uint256 tokenAmount) private {
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 addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
_approve(address(this), address(uniswapV2Router), tokenAmount);

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

function _tokenTransfer(address sender, address recipient, uint256 amount) private {
if(_isExcludedFromFee[sender] || _isExcludedFromFee[recipient]){
removeAllFee();
}else{
if(AutomaticMarketMaker[recipient] || recipient == uniswapV2Pair){
_taxFee = SellTaxFee;
_liquidityFee = SellLiquidityFee;
_CharityFee = SellCharityFee;
```

```
811
812 }else{
813 _taxFee = buyTaxFee;
814 _liquidityFee = buyLiquidityFee;
815 _CharityFee = buyCharityFee;
816 }
817 }
818
819
820 uint256 CharityAmt = amount.mul(_CharityFee).div(100);
821
822
823 if (_isExcluded[sender] && !_isExcluded[recipient]) {
824 _transferFromExcluded(sender, recipient, (amount.sub(CharityAmt)));
825 } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
826 _transferToExcluded(sender, recipient, (amount.sub(CharityAmt)));
827 } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
828 _transferStandard(sender, recipient, (amount.sub(CharityAmt)));
829 } else if (_isExcluded[sender] && _isExcluded[recipient]) {
830 _transferBothExcluded(sender, recipient, (amount.sub(CharityAmt)));
831 } else {
832 _transferStandard(sender, recipient, (amount.sub(CharityAmt)));
833 }
834
835 _taxFee = 0;
836 _liquidityFee = 0;
837
838 _transferStandard(sender, CharityWallet, CharityAmt);
839
840 _taxFee = _previousTaxFee;
841 _liquidityFee = _previousLiquidityFee;
842
843
844 if(_isExcludedFromFee[sender] || _isExcludedFromFee[recipient] || AutomaticMarketMaker[recipient])
845 restoreAllFee();
846
847 }
848
849 function _transferStandard(address sender, address recipient, uint256 tAmount) private {
850 (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getVal
ues(tAmount);
851 _rOwned[sender] = _rOwned[sender].sub(rAmount);
852 _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
853 _takeLiquidity(tLiquidity);
854 _reflectFee(rFee, tFee);
855 emit Transfer(sender, recipient, tTransferAmount);
856 }
857
858 function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
859 (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getVal
ues(tAmount);
860 _rOwned[sender] = _rOwned[sender].sub(rAmount);
861 _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
862 _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
863 _takeLiquidity(tLiquidity);
864 _reflectFee(rFee, tFee);
865 emit Transfer(sender, recipient, tTransferAmount);
866 }
867
868 function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
869 (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getVal
ues(tAmount);
870 _tOwned[sender] = _tOwned[sender].sub(tAmount);
871 _rOwned[sender] = _rOwned[sender].sub(rAmount);
872 _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
873 _takeLiquidity(tLiquidity);
874 _reflectFee(rFee, tFee);
875 emit Transfer(sender, recipient, tTransferAmount);
876 }
877}
```