/\*

#LIQ+#RFI+#SHIB+#DOGE = #GST

#Guardiansale Token features:

Every transaction takes 4% total tax fee,

1% fee to the liquidity pool,

1% fee distributed to all holders,

1% fee for marketing purposes,

1% fee buyback & burn.

there is no auto pre-burn.

\*/

// SPDX-License-Identifier: Unlicensed

// Guardiansale BSC contract

// Deflationary social mining on BSC

pragma solidity >=0.5.16;

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;

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) {

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;

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 () {

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 getUnlockTime() 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 = block.timestamp + 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(block.timestamp > \_lockTime , "Contract is locked until 7 days");

emit OwnershipTransferred(\_owner, \_previousOwner);

\_owner = \_previousOwner;

}

}

// pragma solidity >=0.5.0;

interface IpancakeV2Factory {

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 IpancakeV2Pair {

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 IpancakeV2Router01 {

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);

}

interface IpancakeV2Router02 is IpancakeV2Router01 {

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;

}

// GuardianSale Token contract starts here..

contract GuardianSaleToken is Context, IERC20, Ownable {

using SafeMath for uint256;

using Address for address;

mapping (address => uint256) private \_rOwned; // reflection owned by a certain address.

mapping (address => uint256) private \_tOwned; // tokens owned by a certain address.

mapping (address => mapping (address => uint256)) private \_allowances; // how much an address is allowed to spend on behalf of another.

mapping (address => bool) private \_isExcludedFromFee; // checks if an address is excluded from paying tax fees.

mapping (address => bool) private \_isExcludedFromMaxTxAmount; // checks if an address is excluded from having \_maxTxAmount transaction limitaion.

mapping (address => bool) private \_isExcluded; // checks if an address is \_excluded from receiving reflection rewards. which means not getting a share of the 1% \_taxFee.

address[] private \_excluded; // list of addresses that are excluded from receiving rewards from reflection.

uint256 private MAX = ~uint256(0); /\* maximum value of 2^256-1 = 11579208923731619542357098500868790785326998466564056403945758400791312963993.

to calculate reflection, multiply the number above with your tokens. or use reflectionFromToken function. \*/

uint256 private \_tTotal = 100000 \* 10\*\*6 \* 10\*\*18; // 100000 \* 10\*\*6 is total token supply and 10\*\*18 are 18 decimals.

uint256 private \_rTotal = (MAX - (MAX % \_tTotal)); // total reflection

uint256 private \_tFeeTotal; // total tokens with holders, this helps calculating how much percentage of 1% taxFee each holder gets. depending on how much of this each holders owns.

string private \_name = "Guardiansale Token";

string private \_symbol = "GST";

uint8 private \_decimals = 18; // decimals for tokens are similar to cents for US dollar.

uint256 public \_taxFee = 1; // 1% for all holders.

uint256 private \_previousTaxFee = \_taxFee; // if taxfee changes, this helps resetting it to the previous value.

uint256 public \_liquidityFee = 3; //(1% LP , 1% Buyback, 1% Marketing)

uint256 private \_previousLiquidityFee = \_liquidityFee; // if liquidityFee changes, this helps resetting it to the previous value.

IpancakeV2Router02 public immutable pancakeV2Router; // to use pancakeV2Router contract inside token contract.

address public immutable pancakeV2Pair; // to use pancakeV2Pair contract inside token contract.

address payable public \_timelockContract; // guardiansale timelock contract address.

address payable buyback; // buyback wallet address. 1% of tax is sent to this wallet to buy and burn tokens.

address payable marketing; // marketing wallet address. 1% of tax is collected for marketing the project.

bool inSwapAndLiquify; // it's value, which is true or false, determines locking swaps of tokens for liquifying if swaping is already ongoing.

bool public swapAndLiquifyEnabled = true; // this can be changed later into false if owner wants to disable the swapping event.

uint256 public \_maxTxAmount = 20 \* 10\*\*6 \* 10\*\*18; // maximum tokens that can be transected, this restricts whales from selling all tokens with one transaction.

/\*minTokensBeforeSwap: at each trasnaction 3% fee is charged '\_liquidityFee' and sent to contract balance. When the contract balance reaches the minTokensBeforeSwap amount,

at the next transaction, it will execute the function swapAndLiquify which divides the contract balance to 3. first part goes to liquidity, second to buyback and third to marketing.

you can see the full process explained with more details in swapAndLiquify function\*/

uint256 private constant minTokensBeforeSwap = 50 \* 10\*\*6 \* 10\*\*18;

event SwapAndLiquifyEnabledUpdated(bool enabled); // updates SwapAndLiquifyEnabled

// this event is used in SwapAndLiquify function, you can check the values use and description inside the function.

event SwapAndLiquify(

uint256 tokensSwapped,

uint256 ethReceived,

uint256 tokensIntoLiquidity

);

// locks swap when a swap is already ongoing.

modifier lockTheSwap {

inSwapAndLiquify = true;

\_;

inSwapAndLiquify = false;

}

// Constructor to determine the specific parameters of token contract.

constructor () {

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

IpancakeV2Router02 \_pancakeV2Router = IpancakeV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E); // pancakeV2Router address on BSC mainnet.

pancakeV2Pair = IpancakeV2Factory(\_pancakeV2Router.factory())

.createPair(address(this), \_pancakeV2Router.WETH()); // creating pair address on pancakeswap for the liquidity pool.

pancakeV2Router = \_pancakeV2Router;

buyback = payable(0x1339302129BDF6e358768CCb0b7d84f7ff761529); // buyback wallet.

marketing= payable(0x3B9e1b8056E8E711D665C98c98e294D332553c18); // marketing wallet.

\_isExcludedFromFee[owner()] = true; // owner is excluded from taxfee.

\_isExcludedFromFee[address(this)] = true; // token address is excluded from taxfee.

emit Transfer(address(0), \_msgSender(), \_tTotal); // when contract is created, total tokens are sent to owner address.

}

// returns token name, \*\*public view means anyone can READ this function from outside the contract.

function name() public view returns (string memory) {

return \_name;

}

// returns token symbol.

function symbol() public view returns (string memory) {

return \_symbol;

}

// returns token decimals.

function decimals() public view returns (uint8) {

return \_decimals;

}

// returns total supply of token.

function totalSupply() public view override returns (uint256) {

return \_tTotal;

}

// returns balance of the provided address.

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

if (\_isExcluded[account]) return \_tOwned[account];

return tokenFromReflection(\_rOwned[account]);

}

// transfers tokens from the sender, to a recipient address with certain amount of tokens.

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

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

return true;

}

// returns the remaining number of tokens that `spender` is allowed to spend on behalf of `owner`.

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

return \_allowances[owner][spender];

}

// sets `amount` as the allowance of `spender` over the token owner's tokens.

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

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

return true;

}

// owner sets the timelock address after deployment of contract. note that \*\*onlyOwner() means only owner of token contract is able to use this function.

function setTimelockContract(address payable timelock) external onlyOwner() {

\_timelockContract = timelock;

}

// transfer tokens from an address to another, with certain amount. this function can only work if the caller has allowance from sender to transfer on their behalf.

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;

}

// caller adds more tokens to a spender address allowance.

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

\_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender].add(addedValue));

return true;

}

// caller substracts tokens from a spender address allowance.

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;

}

// checking if an address is excluded from receiving reflection rewards, which comes from the 1% taxfee. returns true if address is excluded, false if not.

function isExcludedFromReward(address account) public view returns (bool) {

return \_isExcluded[account];

}

// returns total tokens owned by holders

function totalFees() public view returns (uint256) {

return \_tFeeTotal;

}

// destroys sender tokens and contribute to other token holders, any token holder that is not excluded from rewards is able to use this function.

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);

}

// calculates reflection by providing tokens amount. you can choose to calculate with or without taxes by setting deductTransferFee to true or false.

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;

}

}

// calculates tokens by providing reflection amount.

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);

}

// owner adds an address to exclude from taking from the 1% reflection taxfee.

function excludeFromReward(address account) public onlyOwner() {

require(!\_isExcluded[account], "Account is already excluded");

if(\_rOwned[account] > 0) {

\_tOwned[account] = tokenFromReflection(\_rOwned[account]);

}

\_isExcluded[account] = true;

\_excluded.push(account);

}

// owner adds an address to include in taking from the 1% reflection taxfee.

function includeInReward(address account) external onlyOwner() {

require(\_isExcluded[account], "Account is not 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;

}

}

}

// owner excludes an address from paying taxes.

function excludeFromFee(address account) public onlyOwner {

\_isExcludedFromFee[account] = true;

}

// owner includes an address to pay taxes.

function includeInFee(address account) public onlyOwner {

\_isExcludedFromFee[account] = false;

}

// owner excludes an address from being restricted with maxTxAmount.

function excludeFromMaxTxAmount(address account) public onlyOwner {

\_isExcludedFromMaxTxAmount[account] = true;

}

// owner includes an address to be restricted with maxTxAmount.

function includeInMaxTxAmount(address account) public onlyOwner {

\_isExcludedFromMaxTxAmount[account] = false;

}

// taxfee is initially 1%, but owner can change it with this function.

function setTaxFeePercent(uint256 taxFee) external onlyOwner() {

\_taxFee = taxFee;

}

// liquidityfee is initially 3%, divided between liquidity, buyback and marketing. the owner can change this percentage.

function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {

\_liquidityFee = liquidityFee;

}

// changes the percentage of \_maxTxAmount to the total supply of token. which means the initial \_maxTxAmount, that is the limit for each transaction can be changed.

function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {

\_maxTxAmount = \_tTotal.mul(maxTxPercent).div(

10\*\*2

);

}

// owner can disable and enable back swapAndLiquify. more details about this in swapAndLiquify function.

function setSwapAndLiquifyEnabled(bool \_enabled) public onlyOwner {

swapAndLiquifyEnabled = \_enabled;

emit SwapAndLiquifyEnabledUpdated(\_enabled);

}

// contract is able to receive BNB from pancakeswap when SwapAndLiquify is excuted.

receive() external payable {}

// substracts from \_rTotal 'reflection' and adds to \_tFeeTotal 'tokens'.

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

\_rTotal = \_rTotal.sub(rFee);

\_tFeeTotal = \_tFeeTotal.add(tFee);

}

// returns reflection and token values from a certain token amount. using \_getTValues for token values 'transferred token, taxfee, liquidityfee'

// and \_getRValues to return reflection values of each token value. please check the next 2 functions as well.

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);

}

// when a token transfer occurs, this function calculates how much is actually transferred after calculating and taking taxfee '1%' and liquidityfee '3%'.

// it returns then all three values, taxfee => tFee, liquidityfee => tliquidity, and the tokens that are actually being transferred after taxes cut.

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);

}

// returns reflection values after calculating according to the rate of reflection to token.

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);

}

// returns the rate between reflection and token by dividing reflection/token.

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

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

return rSupply.div(tSupply);

}

// calculates and returns total reflection, and total token. after substracting tokens and reflection of excluded addresses.

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);

}

// takes liquidityfee that is 3% and adds it to the token contract balance. for it to be divided later on to three, liquidity, buyback and marketing.

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);

}

// calculates taxFee of a certain amount. note that \_taxFee is initially 1% and is shared between token holders.

function calculateTaxFee(uint256 \_amount) private view returns (uint256) {

return \_amount.mul(\_taxFee).div(

10\*\*2

);

}

// calculates liquidityFee of a certain amount. note that liquidityFee is initially 3% and includes buyback and marketing as well.

function calculateLiquidityFee(uint256 \_amount) private view returns (uint256) {

return \_amount.mul(\_liquidityFee).div(

10\*\*2

);

}

// removes all tax fees.

function removeAllFee() private {

if(\_taxFee == 0 && \_liquidityFee == 0) return;

\_previousTaxFee = \_taxFee;

\_previousLiquidityFee = \_liquidityFee;

\_taxFee = 0;

\_liquidityFee = 0;

}

// resets the initial fees for taxfee and liquidity fee, if they were changed.

function restoreAllFee() private {

\_taxFee = \_previousTaxFee;

\_liquidityFee = \_previousLiquidityFee;

}

// you can enter an address and check if it's excluded from paying tax fees or not. it returns true if it is excluded, false if not excluded.

function isExcludedFromFee(address account) public view returns(bool) {

return \_isExcludedFromFee[account];

}

/\* this approve function is different from the other. the first one is used to 'approve' from outside the contract.

while '\_approve' has the approval event as well which means it excutes approval inside the contract.

this is useful with approving Router's address when adding liquidity. without the need of doing that manually

everytime the SwapAndLiquify is invoked. \*/

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);

}

/\* this function handles transfers of token. first it checks several conditions for the transaction to pass, like valid addresses and not passing

the maximum amount allowed for a transfer '\_maxTxAmount'. if the transaction is made from or to an excluded from fee address, transaction will

not take fee. it also checks with every transaction if minTokensBeforeSwap is reached in the contract balance. which grows with each transaction

by taking 3% fees, and when it reaches minTokensBeforeSwap, the next transaction will have SwapAndLiquify function excuted, and fees distributed \*/

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()) && (!\_isExcludedFromMaxTxAmount[from] && !\_isExcludedFromMaxTxAmount[to]))

require(amount <= \_maxTxAmount, "Transfer amount exceeds the maxTxAmount.");

uint256 contractTokenBalance = balanceOf(address(this));

if(contractTokenBalance >= \_maxTxAmount)

{

contractTokenBalance = \_maxTxAmount;

}

bool overMinTokenBalance = contractTokenBalance >= minTokensBeforeSwap;

if (

overMinTokenBalance &&

!inSwapAndLiquify &&

from != pancakeV2Pair &&

swapAndLiquifyEnabled

) {

contractTokenBalance = minTokensBeforeSwap;

swapAndLiquify(contractTokenBalance);

}

bool takeFee = true;

if(\_isExcludedFromFee[from] || \_isExcludedFromFee[to]){

takeFee = false;

}

\_tokenTransfer(from,to,amount,takeFee);

}

/\* with every transaction, the contract token balance is checked until it reaches minTokensBeforeSwap, then, the balance is divided by 3.

one third for liquidity, one for buyback and one for marketing. the one third of liquidity is split in half, half remains as token,

the other half is swapped with BNB using pancakeswap, then create liquidity with addLiquidity function using those two halves. buyback and marketing

are also swapped with BNB then sent to the wallets. \*/

function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {

// contractTokenBalance is divided by 3

uint256 liquiditys = contractTokenBalance.div(3);

uint256 buybacks = contractTokenBalance.div(3);

uint256 marketings = contractTokenBalance.div(3);

// split the liquidity in half

uint256 half = liquiditys.div(2);

// otherhalf of liquiditys after division

uint256 otherHalf = liquiditys.sub(half);

/\*capture the contract's current BNB balance.

this is so that we can capture exactly the amount of BNB that the

swap creates, and not make the liquidity event include any BNB that

has been manually sent to the contract\*/

uint256 initialBalance = address(this).balance;

// swap tokens for BNB,

swapTokensForEth(half.add(buybacks).add(marketings)); // <- this breaks the BNB -> HATE swap when swap+liquify is triggered

// generate the pancakeswap pair path of token -> BNB

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

uint256 oneThird = Balance.div(3);

// buyback & burn fee

buyback.transfer(oneThird);

// marketing fee

marketing.transfer(oneThird);

// add liquidity to SWAP

addLiquidity(otherHalf, oneThird);

emit SwapAndLiquify(half, oneThird, otherHalf);

}

// returns BNB balance of the token contract.

function BNBBalance() external view returns(uint256){

return address(this).balance;

}

// in case any BNB is left from swapAndLiquify or if someone accidently sends BNB to token address, we are sending them to the timelock contract.

function BNBleftover() public virtual onlyOwner {

uint256 Balance = address(this).balance;

\_timelockContract.transfer(Balance);

}

// swapping tokens with BNB from the token's liquidity pool on pancakeswap.

function swapTokensForEth(uint256 tokenAmount) private {

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

path[0] = address(this);

path[1] = pancakeV2Router.WETH();

\_approve(address(this), address(pancakeV2Router), tokenAmount);

// make the swap

pancakeV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(

tokenAmount,

0,

path,

address(this),

block.timestamp

);

}

// add liquidity to pancakeswap liquidity pool.

function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {

// approve token transfer to cover all possible scenarios

\_approve(address(this), address(pancakeV2Router), tokenAmount);

// add the liquidity

pancakeV2Router.addLiquidityETH{value: ethAmount}(

address(this),

tokenAmount,

0,

0,

\_timelockContract, // timelock contract will receive all LP tokens whenever more liquidity is generated by this token contract.

block.timestamp

);

}

// excluded and not excluded in the next functions means, excluded from receiving reflection rewards or not.

// this method is responsible for taking all fees, if takeFee is true. every condition is using the functions following this one.

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();

}

// handles a token transfer between a sender that is not excluded, to a recipient that is not excluded as well.

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);

}

// handles a token transfer between a sender that is not excluded, to a recipient that is excluded.

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);

}

// handles a token transfer between a sender that is excluded, to a recipient that is not excluded.

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);

}

// handles a token transfer between a sender that is excluded, to a recipient that is excluded as well.

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);

}

}