```
/**
 *Submitted for verification at BscScan.com on 2021-11-21
*/

// Dependency file: @openzeppelin/contracts/token/ERC20/IERC20.sol

// SPDX-License-Identifier: MIT

// pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    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);
}


// Dependency file: @openzeppelin/contracts/utils/Context.sol



// pragma solidity ^0.8.0;
```

```
/**
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}



// Dependency file: @openzeppelin/contracts/access/Ownable.sol



// pragma solidity ^0.8.0;

// import "@openzeppelin/contracts/utils/Context.sol";

/**
 * @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.
 */
abstract contract Ownable is Context {
    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        _setOwner(_msgSender());
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual 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 {
        _setOwner(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");
        _setOwner(newOwner);
    }

    function _setOwner(address newOwner) private {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
```

```
}



// Dependency file: @openzeppelin/contracts/utils/math/SafeMath.sol



// pragma solidity ^0.8.0;

// CAUTION
// This version of SafeMath should only be used with Solidity 0.8 or later,
// because it relies on the compiler's built in overflow checks.

/**
 * @dev Wrappers over Solidity's arithmetic operations.
 *
 * NOTE: `SafeMath` is no longer needed starting with Solidity 0.8. The compiler
 * now has built in overflow checking.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }
```

```
/**
 * @dev Returns the substraction of two unsigned integers, with an overflow flag.
 *
 * _Available since v3.4._
 */
function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        if (b > a) return (false, 0);
        return (true, a - b);
    }
}

/**
 * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
 *
 * _Available since v3.4._
 */
function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        // 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 (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }
}

/**
```

```
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a / b);
        }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero
flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a % b);
        }
    }

    /**
     * @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) {
   return a + b;
}

/**
 * @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 a - b;
}

/**
 * @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) {
        return a * b;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator.
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
modulo),
     * reverting 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 a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message
on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b <= a, errorMessage);
            return a - b;
        }
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting 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) {
        unchecked {
            require(b > 0, errorMessage);
            return a / b;
        }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * 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) {
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
}


// Dependency file: @openzeppelin/contracts/utils/Address.sol



// pragma solidity ^0.8.0;

/**
 * @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) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        assembly {
            size := extcodesize(account)
        }
        return size > 0;
    }

    /**
     * @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");

        (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");
        require(isContract(target), "Address: call to non-contract");

        (bool success, bytes memory returndata) = target.call{value: value}(data);
        return verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes
memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        (bool success, bytes memory returndata) = target.staticcall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }

    /**
```

```
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes
memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        (bool success, bytes memory returndata) = target.delegatecall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Tool to verifies that a low level call was successful, and revert if it wasn't, either by
bubbling the
     * revert reason using the provided one.
     *
     * _Available since v4.3._
     */
    function verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) internal pure returns (bytes memory) {
        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

                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}



// Dependency file: contracts/interfaces/IUniswapV2Router02.sol


// 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,
  uint256 amountADesired,
  uint256 amountBDesired,
  uint256 amountAMin,
  uint256 amountBMin,
  address to,
  uint256 deadline
)
  external
  returns (
    uint256 amountA,
    uint256 amountB,
    uint256 liquidity
  );

function addLiquidityETH(
  address token,
  uint256 amountTokenDesired,
  uint256 amountTokenMin,
  uint256 amountETHMin,
  address to,
  uint256 deadline
)
  external
  payable
```

```
    returns (
        uint256 amountToken,
        uint256 amountETH,
        uint256 liquidity
    );

function removeLiquidity(
    address tokenA,
    address tokenB,
    uint256 liquidity,
    uint256 amountAMin,
    uint256 amountBMin,
    address to,
    uint256 deadline
) external returns (uint256 amountA, uint256 amountB);

function removeLiquidityETH(
    address token,
    uint256 liquidity,
    uint256 amountTokenMin,
    uint256 amountETHMin,
    address to,
    uint256 deadline
) external returns (uint256 amountToken, uint256 amountETH);

function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint256 liquidity,
    uint256 amountAMin,
    uint256 amountBMin,
    address to,
    uint256 deadline,
    bool approveMax,
    uint8 v,
    bytes32 r,
    bytes32 s
) external returns (uint256 amountA, uint256 amountB);

function removeLiquidityETHWithPermit(
    address token,
    uint256 liquidity,
    uint256 amountTokenMin,
    uint256 amountETHMin,
    address to,
    uint256 deadline,
    bool approveMax,
    uint8 v,
    bytes32 r,
    bytes32 s
) external returns (uint256 amountToken, uint256 amountETH);

function swapExactTokensForTokens(
    uint256 amountIn,
    uint256 amountOutMin,
    address[] calldata path,
    address to,
    uint256 deadline
) external returns (uint256[] memory amounts);

function swapTokensForExactTokens(
    uint256 amountOut,
    uint256 amountInMax,
    address[] calldata path,
    address to,
    uint256 deadline
) external returns (uint256[] memory amounts);

function swapExactETHForTokens(
    uint256 amountOutMin,
    address[] calldata path,
    address to,
    uint256 deadline
) external payable returns (uint256[] memory amounts);

function swapTokensForExactETH(
    uint256 amountOut,
    uint256 amountInMax,
    address[] calldata path,
    address to,
    uint256 deadline
) external returns (uint256[] memory amounts);

function swapExactTokensForETH(
    uint256 amountIn,
    uint256 amountOutMin,
    address[] calldata path,
    address to,
    uint256 deadline
) external returns (uint256[] memory amounts);

function swapETHForExactTokens(
    uint256 amountOut,
    address[] calldata path,
    address to,
    uint256 deadline
) external payable returns (uint256[] memory amounts);

function quote(
    uint256 amountA,
    uint256 reserveA,
    uint256 reserveB
) external pure returns (uint256 amountB);

function getAmountOut(
    uint256 amountIn,
    uint256 reserveIn,
    uint256 reserveOut
) external pure returns (uint256 amountOut);

function getAmountIn(
    uint256 amountOut,
    uint256 reserveIn,
    uint256 reserveOut
) external pure returns (uint256 amountIn);

function getAmountsOut(uint256 amountIn, address[] calldata path)
    external
    view
    returns (uint256[] memory amounts);

function getAmountsIn(uint256 amountOut, address[] calldata path)
    external
    view
```

```
        returns (uint256[] memory amounts);
}

interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountETH);

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable;

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;
}


// Dependency file: contracts/interfaces/IUniswapV2Factory.sol

// pragma solidity >=0.5.0;

interface IUniswapV2Factory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );
    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(uint256) external view returns (address pair);

    function allPairsLength() external view returns (uint256);

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

    function setFeeTo(address) external;

    function setFeeToSetter(address) external;
}


// Dependency file: contracts/BaseToken.sol

// pragma solidity =0.8.4;

enum TokenType {
    standard,
    antiBotStandard,
    liquidityGenerator,
    antiBotLiquidityGenerator,
    baby,
    antiBotBaby,
    buybackBaby,
    antiBotBuybackBaby
}

abstract contract BaseToken {
    event TokenCreated(
        address indexed owner,
        address indexed token,
        TokenType tokenType,
        uint256 version
    );
}



// Root file: contracts/liquidity-generator/LiquidityGeneratorToken.sol

pragma solidity =0.8.4;

// import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
// import "@openzeppelin/contracts/access/Ownable.sol";
// import "@openzeppelin/contracts/utils/math/SafeMath.sol";
// import "@openzeppelin/contracts/utils/Address.sol";
// import "contracts/interfaces/IUniswapV2Router02.sol";
// import "contracts/interfaces/IUniswapV2Factory.sol";
// import "contracts/BaseToken.sol";

contract LiquidityGeneratorToken is IERC20, Ownable, BaseToken {
```

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

uint256 public constant VERSION = 1;

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;
uint256 private _rTotal;
uint256 private _tFeeTotal;

string private _name;
string private _symbol;
uint8 private _decimals;

uint256 public _taxFee;
uint256 private _previousTaxFee = _taxFee;

uint256 public _liquidityFee;
uint256 private _previousLiquidityFee = _liquidityFee;

uint256 public _charityFee;
uint256 private _previousCharityFee = _charityFee;
```

```
IUniswapV2Router02 public uniswapV2Router;
address public uniswapV2Pair;
address public _charityAddress;

bool inSwapAndLiquify;
bool public swapAndLiquifyEnabled;

uint256 private numTokensSellToAddToLiquidity;

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

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

constructor(
   string memory name_,
   string memory symbol_,
   uint256 totalSupply_,
   address router_,
   address charityAddress_,
   uint16 taxFeeBps_,
   uint16 liquidityFeeBps_,
   uint16 charityFeeBps_,
   address serviceFeeReceiver_,
   uint256 serviceFee_
) payable {
   require(taxFeeBps_ >= 0, "Invalid tax fee");
   require(liquidityFeeBps_ >= 0, "Invalid liquidity fee");
   require(charityFeeBps_ >= 0, "Invalid charity fee");
   if (charityAddress_ == address(0)) {
     require(
       charityFeeBps_ == 0,
       "Cant set both charity address to address 0 and charity percent more than 0"
     );
   }
   require(
     taxFeeBps_ + liquidityFeeBps_ + charityFeeBps_ <= 10**4 / 4,
     "Total fee is over 25%"
   );

   _name = name_;
   _symbol = symbol_;
   _decimals = 9;

   _tTotal = totalSupply_;
   _rTotal = (MAX - (MAX % _tTotal));

   _taxFee = taxFeeBps_;
   _previousTaxFee = _taxFee;

   _liquidityFee = liquidityFeeBps_;
   _previousLiquidityFee = _liquidityFee;
```

```
_charityAddress = charityAddress_;
_charityFee = charityFeeBps_;
_previousCharityFee = _charityFee;

numTokensSellToAddToLiquidity = totalSupply_.mul(5).div(10**4); // 0.05%

swapAndLiquifyEnabled = true;

_rOwned[owner()] = _rTotal;

IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(router_);
// 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), owner(), _tTotal);

emit TokenCreated(
    owner(),
    address(this),
    TokenType.liquidityGenerator,
    VERSION
);
```

```
    payable(serviceFeeReceiver_).transfer(serviceFee_);
}

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,
        uint256 tCharity
    ) = _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);
    _takeCharityFee(tCharity);
    _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 taxFeeBps) external onlyOwner {
    _taxFee = taxFeeBps;
    require(
        _taxFee + _liquidityFee + _charityFee <= 10**4 / 4,
        "Total fee is over 25%"
    );
}

function setLiquidityFeePercent(uint256 liquidityFeeBps)
    external
    onlyOwner
{
    _liquidityFee = liquidityFeeBps;
    require(
        _taxFee + _liquidityFee + _charityFee <= 10**4 / 4,
        "Total fee is over 25%"
    );
}

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
    )
{
    (
        uint256 tTransferAmount,
        uint256 tFee,
        uint256 tLiquidity,
        uint256 tCharity
    ) = _getTValues(tAmount);
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(
        tAmount,
        tFee,
        tLiquidity,
        tCharity,
        _getRate()
    );
    return (
        rAmount,
        rTransferAmount,
        rFee,
        tTransferAmount,
        tFee,
        tLiquidity,
        tCharity
    );
}

function _getTValues(uint256 tAmount)
    private
    view
    returns (
        uint256,
        uint256,
        uint256,
        uint256
    )
```

```
{
   uint256 tFee = calculateTaxFee(tAmount);
   uint256 tLiquidity = calculateLiquidityFee(tAmount);
   uint256 tCharityFee = calculateCharityFee(tAmount);
   uint256 tTransferAmount = tAmount.sub(tFee).sub(tLiquidity).sub(
      tCharityFee
   );
   return (tTransferAmount, tFee, tLiquidity, tCharityFee);
}

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

```
  );
  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 _takeCharityFee(uint256 tCharity) private {
    if (tCharity > 0) {
        uint256 currentRate = _getRate();
        uint256 rCharity = tCharity.mul(currentRate);
        _rOwned[_charityAddress] = _rOwned[_charityAddress].add(rCharity);
        if (_isExcluded[_charityAddress])
            _tOwned[_charityAddress] = _tOwned[_charityAddress].add(
                tCharity
            );
        emit Transfer(_msgSender(), _charityAddress, tCharity);
    }
}

function calculateTaxFee(uint256 _amount) private view returns (uint256) {
    return _amount.mul(_taxFee).div(10**4);
}

function calculateLiquidityFee(uint256 _amount)
    private
    view
    returns (uint256)
{
    return _amount.mul(_liquidityFee).div(10**4);
}

function calculateCharityFee(uint256 _amount)
    private
    view
    returns (uint256)
{
    if (_charityAddress == address(0)) return 0;
    return _amount.mul(_charityFee).div(10**4);
}

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

    _previousTaxFee = _taxFee;
    _previousLiquidityFee = _liquidityFee;
    _previousCharityFee = _charityFee;

    _taxFee = 0;
    _liquidityFee = 0;
    _charityFee = 0;
}

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

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");

    // 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));

    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,
            uint256 tCharity
        ) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _takeCharityFee(tCharity);
        _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,
            uint256 tCharity
        ) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _takeCharityFee(tCharity);
        _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,
            uint256 tCharity
        ) = _getValues(tAmount);
        _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _takeCharityFee(tCharity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }
}
```