- 24 Removals + 7 Additions

```
/**
 *Submitted for verification at BscScan.com on 2021-04-24
*/

/**
    #MARIOBROS COIN


    #MARIO features:
   7% fee auto add to the liquidity pool to locked forever when selling
   3% fee auto distribute to all holders
   I created a black hole so #MARIO token will deflate itself in supply with every transaction
   25% Supply is burned at start.
   also there is antiwhale system every buy and sell

   I will add 0.3 bnb as an initial liquidity, and burn 25% from the start to create the blackhole.
   0.05% team token, and after that i will burn the LP and renounce Ownership
   can u make #MARIO 100000x???
   i will give this token to the community, please make telegram t.me/MARIOBROS_COIN
 */

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

    function totalSupply() external view returns (uint256);

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
```

```
/**
 *Submitted for verification at BscScan.com on 2021-04-05
*/

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

    function totalSupply() external view returns (uint256);

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

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

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
```

```
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

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

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

```
     * 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).
```

```
}



/**
 * @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
```

```
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

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

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

        return c;
    }

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

```
zero");
    }

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

        return c;
    }

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

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
```

```
zero");
    }

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

        return c;
    }

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

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
```

```
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

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

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


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

```
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

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

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


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

```
cak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

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

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
```

```
cak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

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

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
```

```
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

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

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
```

```
     *
     * 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.
 */
```

```
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

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

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}

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

```
contract Ownable is Context {
    address private _owner;
    address private _previousOwner;
    uint256 private _lockTime;

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

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

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

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

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

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

```
contract Ownable is Context {
    address private _owner;
    address private _previousOwner;
    uint256 private _lockTime;

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

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

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

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

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

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

```
        _owner = newOwner;
    }

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

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

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

// pragma solidity >=0.5.0;

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

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

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

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

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


// pragma solidity >=0.5.0;

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

    function name() external pure returns (string memory);
```

```
        _owner = newOwner;
    }

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

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

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

// pragma solidity >=0.5.0;

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

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

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

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

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


// pragma solidity >=0.5.0;

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

    function name() external pure returns (string memory);
```

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

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

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

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

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

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

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1
```

```
    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 amount1
```

```
Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

// pragma solidity >=0.6.2;

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

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amou
```

```
Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

// pragma solidity >=0.6.2;

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

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amou
```

```
ntB);
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, by
tes32 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 amountO
utMin, address[] calldata path, address to, uin
t deadline)
        external
        payable
        returns (uint[] memory amounts);
    function swapTokensForExactETH(uint amountO
ut, uint amountInMax, address[] calldata path,
 address to, uint deadline)
        external
        returns (uint[] memory amounts);
    function swapExactTokensForETH(uint amountI
n, uint amountOutMin, address[] calldata path,
 address to, uint deadline)
        external
        returns (uint[] memory amounts);
    function swapETHForExactTokens(uint amountO
ut, address[] calldata path, address to, uint d
eadline)
        external
        payable
        returns (uint[] memory amounts);

    function quote(uint amountA, uint reserveA,
uint reserveB) external pure returns (uint amou
ntB);
    function getAmountOut(uint amountIn, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountOut);
    function getAmountIn(uint amountOut, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountIn);
    function getAmountsOut(uint amountIn, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
    function getAmountsIn(uint amountOut, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
}

```

```
ntB);
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, by
tes32 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 amountO
utMin, address[] calldata path, address to, uin
t deadline)
        external
        payable
        returns (uint[] memory amounts);
    function swapTokensForExactETH(uint amountO
ut, uint amountInMax, address[] calldata path,
 address to, uint deadline)
        external
        returns (uint[] memory amounts);
    function swapExactTokensForETH(uint amountI
n, uint amountOutMin, address[] calldata path,
 address to, uint deadline)
        external
        returns (uint[] memory amounts);
    function swapETHForExactTokens(uint amountO
ut, address[] calldata path, address to, uint d
eadline)
        external
        payable
        returns (uint[] memory amounts);

    function quote(uint amountA, uint reserveA,
uint reserveB) external pure returns (uint amou
ntB);
    function getAmountOut(uint amountIn, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountOut);
    function getAmountIn(uint amountOut, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountIn);
    function getAmountsOut(uint amountIn, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
    function getAmountsIn(uint amountOut, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
}

```

```
656
657
658 // pragma solidity >=0.6.2;
659
660 interface IUniswapV2Router02 is IUniswapV2Route
    r01 {
661     function removeLiquidityETHSupportingFeeOnT
    ransferTokens(
662         address token,
663         uint liquidity,
664         uint amountTokenMin,
665         uint amountETHMin,
666         address to,
667         uint deadline
668     ) external returns (uint amountETH);
669     function removeLiquidityETHWithPermitSuppor
    tingFeeOnTransferTokens(
670         address token,
671         uint liquidity,
672         uint amountTokenMin,
673         uint amountETHMin,
674         address to,
675         uint deadline,
676         bool approveMax, uint8 v, bytes32 r, by
    tes32 s
677     ) external returns (uint amountETH);
678
679     function swapExactTokensForTokensSupporting
    FeeOnTransferTokens(
680         uint amountIn,
681         uint amountOutMin,
682         address[] calldata path,
683         address to,
684         uint deadline
685     ) external;
686     function swapExactETHForTokensSupportingFee
    OnTransferTokens(
687         uint amountOutMin,
688         address[] calldata path,
689         address to,
690         uint deadline
691     ) external payable;
692     function swapExactTokensForETHSupportingFee
    OnTransferTokens(
693         uint amountIn,
694         uint amountOutMin,
695         address[] calldata path,
696         address to,
697         uint deadline
698     ) external;
699 }
700
701
702 contract MARIOBROS is Context, IERC20, Ownable
     {
703     using SafeMath for uint256;
704     using Address for address;
705
706     mapping (address => uint256) private _rOwne
    d;
707     mapping (address => uint256) private _tOwne
    d;
708     mapping (address => mapping (address => uin
    t256)) private _allowances;
709
710     mapping (address => bool) private _isExclud
    edFromFee;
```

```
639
640
641 // pragma solidity >=0.6.2;
642
643 interface IUniswapV2Router02 is IUniswapV2Route
    r01 {
644     function removeLiquidityETHSupportingFeeOnT
    ransferTokens(
645         address token,
646         uint liquidity,
647         uint amountTokenMin,
648         uint amountETHMin,
649         address to,
650         uint deadline
651     ) external returns (uint amountETH);
652     function removeLiquidityETHWithPermitSuppor
    tingFeeOnTransferTokens(
653         address token,
654         uint liquidity,
655         uint amountTokenMin,
656         uint amountETHMin,
657         address to,
658         uint deadline,
659         bool approveMax, uint8 v, bytes32 r, by
    tes32 s
660     ) external returns (uint amountETH);
661
662     function swapExactTokensForTokensSupporting
    FeeOnTransferTokens(
663         uint amountIn,
664         uint amountOutMin,
665         address[] calldata path,
666         address to,
667         uint deadline
668     ) external;
669     function swapExactETHForTokensSupportingFee
    OnTransferTokens(
670         uint amountOutMin,
671         address[] calldata path,
672         address to,
673         uint deadline
674     ) external payable;
675     function swapExactTokensForETHSupportingFee
    OnTransferTokens(
676         uint amountIn,
677         uint amountOutMin,
678         address[] calldata path,
679         address to,
680         uint deadline
681     ) external;
682 }
683
684
685 contract SafeLight is Context, IERC20, Ownable
     {
686     using SafeMath for uint256;
687     using Address for address;
688
689     mapping (address => uint256) private _rOwne
    d;
690     mapping (address => uint256) private _tOwne
    d;
691     mapping (address => mapping (address => uin
    t256)) private _allowances;
692
693     mapping (address => bool) private _isExclud
    edFromFee;
```

```
711
712     mapping (address => bool) private _isExclud
    ed;
713     address[] private _excluded;
714
715     uint256 private constant MAX = ~uint256(0);
716     uint256 private _tTotal = 1000000000 * 10**
    6 * 10**9;
717     uint256 private _rTotal = (MAX - (MAX % _tT
    otal));
718     uint256 private _tFeeTotal;
719
720     string private _name = "MARIOBROS";
721     string private _symbol = "MARIO";
722     uint8 private _decimals = 9;
723
724     uint256 public _taxFee = 3;
725     uint256 private _previousTaxFee = _taxFee;
726
727     uint256 public _liquidityFee = 7;
728     uint256 private _previousLiquidityFee = _li
    quidityFee;
729
730     IUniswapV2Router02 public immutable uniswap
    V2Router;
731     address public immutable uniswapV2Pair;
732
733     bool inSwapAndLiquify;
734     bool public swapAndLiquifyEnabled = true;
735
736     uint256 public _maxTxAmount = 5000000 * 10*
    *6 * 10**9;
737     uint256 private numTokensSellToAddToLiquidi
    ty = 500000 * 10**6 * 10**9;
738
739     event MinTokensBeforeSwapUpdated(uint256 mi
    nTokensBeforeSwap);
740     event SwapAndLiquifyEnabledUpdated(bool ena
    bled);
741     event SwapAndLiquify(
742         uint256 tokensSwapped,
743         uint256 ethReceived,
744         uint256 tokensIntoLiqudity
745     );
746
747     modifier lockTheSwap {
748         inSwapAndLiquify = true;
749         _;
750         inSwapAndLiquify = false;
751     }
752
753     constructor () public {
754         _rOwned[_msgSender()] = _rTotal;
755
756         IUniswapV2Router02 _uniswapV2Router = I
    UniswapV2Router02(0x05fF2B0DB69458A0750badebc4f
    9e13aDd608C7F);
757          // Create a uniswap pair for this new
     token
758         uniswapV2Pair = IUniswapV2Factory(_unis
    wapV2Router.factory())
759             .createPair(address(this), _uniswap
    V2Router.WETH());
760
761         // set the rest of the contract variabl
    es
```

```
694
695     mapping (address => bool) private _isExclud
    ed;
696     address[] private _excluded;
697
698     uint256 private constant MAX = ~uint256(0);
699     uint256 private _tTotal = 1000000000 * 10**
    6 * 10**9;
700     uint256 private _rTotal = (MAX - (MAX % _tT
    otal));
701     uint256 private _tFeeTotal;
702
703     string private _name = "SafeLight";
704     string private _symbol = "SAFELIGHT";
705     uint8 private _decimals = 9;
706
707     uint256 public _taxFee = 5;
708     uint256 private _previousTaxFee = _taxFee;
709
710     uint256 public _liquidityFee = 5;
711     uint256 private _previousLiquidityFee = _li
    quidityFee;
712
713     IUniswapV2Router02 public immutable uniswap
    V2Router;
714     address public immutable uniswapV2Pair;
715
716     bool inSwapAndLiquify;
717     bool public swapAndLiquifyEnabled = true;
718
719     uint256 public _maxTxAmount = 5000000 * 10*
    *6 * 10**9;
720     uint256 private numTokensSellToAddToLiquidi
    ty = 500000 * 10**6 * 10**9;
721
722     event MinTokensBeforeSwapUpdated(uint256 mi
    nTokensBeforeSwap);
723     event SwapAndLiquifyEnabledUpdated(bool ena
    bled);
724     event SwapAndLiquify(
725         uint256 tokensSwapped,
726         uint256 ethReceived,
727         uint256 tokensIntoLiqudity
728     );
729
730     modifier lockTheSwap {
731         inSwapAndLiquify = true;
732         _;
733         inSwapAndLiquify = false;
734     }
735
736     constructor () public {
737         _rOwned[_msgSender()] = _rTotal;
738
739         IUniswapV2Router02 _uniswapV2Router = I
    UniswapV2Router02(0x05fF2B0DB69458A0750badebc4f
    9e13aDd608C7F);
740          // Create a uniswap pair for this new
     token
741         uniswapV2Pair = IUniswapV2Factory(_unis
    wapV2Router.factory())
742             .createPair(address(this), _uniswap
    V2Router.WETH());
743
744         // set the rest of the contract variabl
    es
```

```
        uniswapV2Router = _uniswapV2Router;

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

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

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }

    function totalSupply() public view override returns (uint256) {
        return _tTotal;
    }

    function balanceOf(address account) public view override returns (uint256) {
        if (_isExcluded[account]) return _tOwned[account];
        return tokenFromReflection(_rOwned[account]);
    }

    function transfer(address recipient, uint256 amount) public override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address owner, address spender) public view override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));

```

```
        uniswapV2Router = _uniswapV2Router;

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

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

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }

    function totalSupply() public view override returns (uint256) {
        return _tTotal;
    }

    function balanceOf(address account) public view override returns (uint256) {
        if (_isExcluded[account]) return _tOwned[account];
        return tokenFromReflection(_rOwned[account]);
    }

    function transfer(address recipient, uint256 amount) public override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address owner, address spender) public view override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));

```

```
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

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

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

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

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

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

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

    function excludeFromReward(address account)
```

```
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

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

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

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

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

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

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

    function excludeFromReward(address account)
```

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

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

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

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

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

```

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

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

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

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

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

```

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

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

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

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

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

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

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

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

```

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

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

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

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

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

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

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

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

```

```
        return (rAmount, rTransferAmount, rFe
e);
    }

    function _getRate() private view returns(ui
nt256) {
        (uint256 rSupply, uint256 tSupply) = _g
etCurrentSupply();
        return rSupply.div(tSupply);
    }

    function _getCurrentSupply() private view r
eturns(uint256, uint256) {
        uint256 rSupply = _rTotal;
        uint256 tSupply = _tTotal;
        for (uint256 i = 0; i < _excluded.lengt
h; i++) {
            if (_rOwned[_excluded[i]] > rSupply
|| _tOwned[_excluded[i]] > tSupply) return (_rT
otal, _tTotal);
            rSupply = rSupply.sub(_rOwned[_excl
uded[i]]);
            tSupply = tSupply.sub(_tOwned[_excl
uded[i]]);
        }
        if (rSupply < _rTotal.div(_tTotal)) ret
urn (_rTotal, _tTotal);
        return (rSupply, tSupply);
    }

    function _takeLiquidity(uint256 tLiquidity)
private {
        uint256 currentRate =  _getRate();
        uint256 rLiquidity = tLiquidity.mul(cur
rentRate);
        _rOwned[address(this)] = _rOwned[addres
s(this)].add(rLiquidity);
        if(_isExcluded[address(this)])
            _tOwned[address(this)] = _tOwned[ad
dress(this)].add(tLiquidity);
    }

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

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

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

        _previousTaxFee = _taxFee;
        _previousLiquidityFee = _liquidityFee;

        _taxFee = 0;
        _liquidityFee = 0;
    }
```

```
        return (rAmount, rTransferAmount, rFe
e);
    }

    function _getRate() private view returns(ui
nt256) {
        (uint256 rSupply, uint256 tSupply) = _g
etCurrentSupply();
        return rSupply.div(tSupply);
    }

    function _getCurrentSupply() private view r
eturns(uint256, uint256) {
        uint256 rSupply = _rTotal;
        uint256 tSupply = _tTotal;
        for (uint256 i = 0; i < _excluded.lengt
h; i++) {
            if (_rOwned[_excluded[i]] > rSupply
|| _tOwned[_excluded[i]] > tSupply) return (_rT
otal, _tTotal);
            rSupply = rSupply.sub(_rOwned[_excl
uded[i]]);
            tSupply = tSupply.sub(_tOwned[_excl
uded[i]]);
        }
        if (rSupply < _rTotal.div(_tTotal)) ret
urn (_rTotal, _tTotal);
        return (rSupply, tSupply);
    }

    function _takeLiquidity(uint256 tLiquidity)
private {
        uint256 currentRate =  _getRate();
        uint256 rLiquidity = tLiquidity.mul(cur
rentRate);
        _rOwned[address(this)] = _rOwned[addres
s(this)].add(rLiquidity);
        if(_isExcluded[address(this)])
            _tOwned[address(this)] = _tOwned[ad
dress(this)].add(tLiquidity);
    }

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

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

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

        _previousTaxFee = _taxFee;
        _previousLiquidityFee = _liquidityFee;

        _taxFee = 0;
        _liquidityFee = 0;
    }
```

```
991
992     function restoreAllFee() private {
993         _taxFee = _previousTaxFee;
994         _liquidityFee = _previousLiquidityFee;
995     }
996
997     function isExcludedFromFee(address account)
    public view returns(bool) {
998         return _isExcludedFromFee[account];
999     }
1000
1001    function _approve(address owner, address sp
    ender, uint256 amount) private {
1002        require(owner != address(0), "ERC20: ap
    prove from the zero address");
1003        require(spender != address(0), "ERC20:
     approve to the zero address");
1004
1005        _allowances[owner][spender] = amount;
1006        emit Approval(owner, spender, amount);
1007    }
1008
1009    function _transfer(
1010        address from,
1011        address to,
1012        uint256 amount
1013    ) private {
1014        require(from != address(0), "ERC20: tra
    nsfer from the zero address");
1015        require(to != address(0), "ERC20: trans
    fer to the zero address");
1016        require(amount > 0, "Transfer amount mu
    st be greater than zero");
1017        if(from != owner() && to != owner())
1018            require(amount <= _maxTxAmount, "Tr
    ansfer amount exceeds the maxTxAmount.");
1019
1020        // is the token balance of this contrac
    t address over the min number of
1021        // tokens that we need to initiate a sw
    ap + liquidity lock?
1022        // also, don't get caught in a circular
    liquidity event.
1023        // also, don't swap & liquify if sender
    is uniswap pair.
1024        uint256 contractTokenBalance = balanceO
    f(address(this));
1025
1026        if(contractTokenBalance >= _maxTxAmoun
    t)
1027        {
1028            contractTokenBalance = _maxTxAmoun
    t;
1029        }
1030
1031        bool overMinTokenBalance = contractToke
    nBalance >= numTokensSellToAddToLiquidity;
1032        if (
1033            overMinTokenBalance &&
1034            !inSwapAndLiquify &&
1035            from != uniswapV2Pair &&
1036            swapAndLiquifyEnabled
1037        ) {
1038            contractTokenBalance = numTokensSel
    lToAddToLiquidity;
1039            //add liquidity
```

```
974
975     function restoreAllFee() private {
976         _taxFee = _previousTaxFee;
977         _liquidityFee = _previousLiquidityFee;
978     }
979
980     function isExcludedFromFee(address account)
    public view returns(bool) {
981         return _isExcludedFromFee[account];
982     }
983
984     function _approve(address owner, address sp
    ender, uint256 amount) private {
985         require(owner != address(0), "ERC20: ap
    prove from the zero address");
986         require(spender != address(0), "ERC20:
     approve to the zero address");
987
988         _allowances[owner][spender] = amount;
989         emit Approval(owner, spender, amount);
990     }
991
992     function _transfer(
993         address from,
994         address to,
995         uint256 amount
996     ) private {
997         require(from != address(0), "ERC20: tra
    nsfer from the zero address");
998         require(to != address(0), "ERC20: trans
    fer to the zero address");
999         require(amount > 0, "Transfer amount mu
    st be greater than zero");
1000        if(from != owner() && to != owner())
1001            require(amount <= _maxTxAmount, "Tr
    ansfer amount exceeds the maxTxAmount.");
1002
1003        // is the token balance of this contrac
    t address over the min number of
1004        // tokens that we need to initiate a sw
    ap + liquidity lock?
1005        // also, don't get caught in a circular
    liquidity event.
1006        // also, don't swap & liquify if sender
    is uniswap pair.
1007        uint256 contractTokenBalance = balanceO
    f(address(this));
1008
1009        if(contractTokenBalance >= _maxTxAmoun
    t)
1010        {
1011            contractTokenBalance = _maxTxAmoun
    t;
1012        }
1013
1014        bool overMinTokenBalance = contractToke
    nBalance >= numTokensSellToAddToLiquidity;
1015        if (
1016            overMinTokenBalance &&
1017            !inSwapAndLiquify &&
1018            from != uniswapV2Pair &&
1019            swapAndLiquifyEnabled
1020        ) {
1021            contractTokenBalance = numTokensSel
    lToAddToLiquidity;
1022            //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);
```

```
            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);
```

```
1085
1086        // make the swap
1087        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
1088            tokenAmount,
1089            0, // accept any amount of ETH
1090            path,
1091            address(this),
1092            block.timestamp
1093        );
1094    }
1095
1096    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
1097        // approve token transfer to cover all possible scenarios
1098        _approve(address(this), address(uniswapV2Router), tokenAmount);
1099
1100        // add the liquidity
1101        uniswapV2Router.addLiquidityETH{value: ethAmount}(
1102            address(this),
1103            tokenAmount,
1104            0, // slippage is unavoidable
1105            0, // slippage is unavoidable
1106            owner(),
1107            block.timestamp
1108        );
1109    }
1110
1111    //this method is responsible for taking all fee, if takeFee is true
1112    function _tokenTransfer(address sender, address recipient, uint256 amount,bool takeFee) private {
1113        if(!takeFee)
1114            removeAllFee();
1115
1116        if (_isExcluded[sender] && !_isExcluded[recipient]) {
1117            _transferFromExcluded(sender, recipient, amount);
1118        } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
1119            _transferToExcluded(sender, recipient, amount);
1120        } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
1121            _transferStandard(sender, recipient, amount);
1122        } else if (_isExcluded[sender] && _isExcluded[recipient]) {
1123            _transferBothExcluded(sender, recipient, amount);
1124        } else {
1125            _transferStandard(sender, recipient, amount);
1126        }
1127
1128        if(!takeFee)
1129            restoreAllFee();
1130    }
1131
1132    function _transferStandard(address sender, address recipient, uint256 tAmount) private {
1133
```

```
1068
1069        // make the swap
1070        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
1071            tokenAmount,
1072            0, // accept any amount of ETH
1073            path,
1074            address(this),
1075            block.timestamp
1076        );
1077    }
1078
1079    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
1080        // approve token transfer to cover all possible scenarios
1081        _approve(address(this), address(uniswapV2Router), tokenAmount);
1082
1083        // add the liquidity
1084        uniswapV2Router.addLiquidityETH{value: ethAmount}(
1085            address(this),
1086            tokenAmount,
1087            0, // slippage is unavoidable
1088            0, // slippage is unavoidable
1089            owner(),
1090            block.timestamp
1091        );
1092    }
1093
1094    //this method is responsible for taking all fee, if takeFee is true
1095    function _tokenTransfer(address sender, address recipient, uint256 amount,bool takeFee) private {
1096        if(!takeFee)
1097            removeAllFee();
1098
1099        if (_isExcluded[sender] && !_isExcluded[recipient]) {
1100            _transferFromExcluded(sender, recipient, amount);
1101        } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
1102            _transferToExcluded(sender, recipient, amount);
1103        } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
1104            _transferStandard(sender, recipient, amount);
1105        } else if (_isExcluded[sender] && _isExcluded[recipient]) {
1106            _transferBothExcluded(sender, recipient, amount);
1107        } else {
1108            _transferStandard(sender, recipient, amount);
1109        }
1110
1111        if(!takeFee)
1112            restoreAllFee();
1113    }
1114
1115    function _transferStandard(address sender, address recipient, uint256 tAmount) private {
1116
```

```
        (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
        _rOwned[sender] = _rOwned[sender].sub(r
Amount);
        _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTrans
ferAmount);
    }

    function _transferToExcluded(address sende
r, address recipient, uint256 tAmount) private
 {
        (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
        _rOwned[sender] = _rOwned[sender].sub(r
Amount);
        _tOwned[recipient] = _tOwned[recipien
t].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTrans
ferAmount);
    }

    function _transferFromExcluded(address send
er, address recipient, uint256 tAmount) private
{
        (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
        _tOwned[sender] = _tOwned[sender].sub(t
Amount);
        _rOwned[sender] = _rOwned[sender].sub(r
Amount);
        _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTrans
ferAmount);
    }



}
```

```
        (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
        _rOwned[sender] = _rOwned[sender].sub(r
Amount);
        _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTrans
ferAmount);
    }

    function _transferToExcluded(address sende
r, address recipient, uint256 tAmount) private
 {
        (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
        _rOwned[sender] = _rOwned[sender].sub(r
Amount);
        _tOwned[recipient] = _tOwned[recipien
t].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTrans
ferAmount);
    }

    function _transferFromExcluded(address send
er, address recipient, uint256 tAmount) private
{
        (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
        _tOwned[sender] = _tOwned[sender].sub(t
Amount);
        _rOwned[sender] = _rOwned[sender].sub(r
Amount);
        _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTrans
ferAmount);
    }



}
```