```
/*



/ $$   / $$                                        | $$      / $$   | $$  /$$
| $$   | $$                                        | $$      | $$   | $$ /$$$$
| $$   | $$ /$$$$$$   /$$$$$$$ /$$$$$$$  /$$$$$$ | $$      | $$   | $$|_  $$
|  $$ / $$//$$__  $$ /$$_____//$$_____/ /$$__  $$| $$      |  $$ / $$/  | $$
 \  $$ $$/| $$$$$$$$|  $$$$$$|  $$$$$$ | $$$$$$$$| $$       \  $$ $$/   | $$
  \  $$$/ | $$_____/ \____  $$\____  $$| $$_____/| $$        \  $$$/    | $$
   \  $/  |  $$$$$$$ /$$$$$$$//$$$$$$$/|  $$$$$$$| $$         \  $/    /$$$$$$
    \_/    \_______/|_______/|_______/  \_______/|__/          \_/    |______/



*/

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

interface IERC20 {

   function totalSupply() external view returns (uint256);

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

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

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

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

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

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

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

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

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

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

       return c;
   }

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

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

       return c;
   }
```

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

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

        return c;
    }

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

```
   }

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

       return c;
   }

   /**
    * @dev Returns the remainder of dividing two unsigned
integers. (unsigned integer modulo),
    * Reverts when dividing by zero.
    *
    * Counterpart to Solidity's `%` operator. This function uses
a `revert`
    * opcode (which leaves remaining gas untouched) while
Solidity uses an
    * invalid opcode to revert (consuming all remaining gas).
    *
    * Requirements:
    *
    * - The divisor cannot be zero.
    */
   function mod(uint256 a, uint256 b) internal pure returns
(uint256) {
       return mod(a, b, "SafeMath: modulo by zero");
   }
   /**
    * @dev Returns the remainder of dividing two unsigned
integers. (unsigned integer modulo),
    * Reverts with custom message when dividing by zero.
    *
    * Counterpart to Solidity's `%` operator. This function uses
a `revert`
    * opcode (which leaves remaining gas untouched) while
Solidity uses an
    * invalid opcode to revert (consuming all remaining gas).
    *
    * Requirements:
    *
    * - The divisor cannot be zero.
    */
   function mod(uint256 a, uint256 b, string memory errorMessage)
internal pure returns (uint256) {
       require(b != 0, errorMessage);
       return a % b;
   }
}

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

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


/**
* @dev Collection of functions related to the address type
*/
library Address {
   /**
    * @dev Returns true if `account` is a contract.
    *
    * [IMPORTANT]
    * ====
    * It is unsafe to assume that an address for which this
function returns
    * false is an externally-owned account (EOA) and not a
contract.
    *
    * Among others, `isContract` will return false for the
following
    * types of addresses:
    *
    *  - an externally-owned account
    *  - a contract in construction
    *  - an address where a contract will be created
    *  - an address where a contract lived, but was destroyed
    * ====
    */
   function isContract(address account) internal view returns
(bool) {
       // According to EIP-1052, 0x0 is the value returned for
not-yet created accounts
       // and
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a47
0 is returned
       // for accounts without code, i.e. `keccak256('')`
       bytes32 codehash;
       bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a47
0;
       // solhint-disable-next-line no-inline-assembly
       assembly { codehash := extcodehash(account) }
       return (codehash != accountHash && codehash != 0x0);
   }

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

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

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

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

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

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

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

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

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

/**
* @dev Contract module which provides a basic access control
mechanism, where
* there is an account (an owner) that can be granted exclusive
access to
* specific functions.
*
* By default, the owner account will be the one that deploys the
```

```
contract. This
* can later be changed with {transferOwnership}.
*
* This module is used through inheritance. It will make available
the modifier
* `onlyOwner`, which can be applied to your functions to restrict
their use to
* the owner.
*/
contract Ownable is Context {
   address private _owner;
   address private _previousOwner;
   uint256 private _lockTime;

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

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

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

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

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

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

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

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

```
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;
}

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

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

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

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

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

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

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

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

   function initialize(address, address) external;
}

library UniswapV2Library {
   using SafeMath for uint;

   // returns sorted token addresses, used to handle return
values from pairs sorted in this order
   function sortTokens(address tokenA, address tokenB) internal
pure returns (address token0, address token1) {
       require(tokenA != tokenB, 'UniswapV2Library:
IDENTICAL_ADDRESSES');
       (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) :
```

```
(tokenB, tokenA);
       require(token0 != address(0), 'UniswapV2Library:
ZERO_ADDRESS');
   }

   // calculates the CREATE2 address for a pair without making
any external calls
   function pairFor(address factory, address tokenA, address
tokenB) internal pure returns (address pair) {
       (address token0, address token1) = sortTokens(tokenA,
tokenB);
       pair = address(uint(keccak256(abi.encodePacked(
               hex'ff',
               factory,
               keccak256(abi.encodePacked(token0, token1)),

hex'96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da3488
45f' // init code hash
           ))));
   }

   // fetches and sorts the reserves for a pair
   function getReserves(address factory, address tokenA, address
tokenB) internal view returns (uint reserveA, uint reserveB) {
       (address token0,) = sortTokens(tokenA, tokenB);
       (uint reserve0, uint reserve1,) =
IUniswapV2Pair(pairFor(factory, tokenA, tokenB)).getReserves();
       (reserveA, reserveB) = tokenA == token0 ? (reserve0,
reserve1) : (reserve1, reserve0);
   }

   // given some amount of an asset and pair reserves, returns an
equivalent amount of the other asset
   function quote(uint amountA, uint reserveA, uint reserveB)
internal pure returns (uint amountB) {
       require(amountA > 0, 'UniswapV2Library:
INSUFFICIENT_AMOUNT');
       require(reserveA > 0 && reserveB > 0, 'UniswapV2Library:
INSUFFICIENT_LIQUIDITY');
       amountB = amountA.mul(reserveB) / reserveA;
   }

   // given an input amount of an asset and pair reserves,
returns the maximum output amount of the other asset
   function getAmountOut(uint amountIn, uint reserveIn, uint
reserveOut) internal pure returns (uint amountOut) {
       require(amountIn > 0, 'UniswapV2Library:
```

```
INSUFFICIENT_INPUT_AMOUNT');
       require(reserveIn > 0 && reserveOut > 0,
'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
       uint amountInWithFee = amountIn.mul(997);
       uint numerator = amountInWithFee.mul(reserveOut);
       uint denominator =
reserveIn.mul(1000).add(amountInWithFee);
       amountOut = numerator / denominator;
   }

   // given an output amount of an asset and pair reserves,
returns a required input amount of the other asset
   function getAmountIn(uint amountOut, uint reserveIn, uint
reserveOut) internal pure returns (uint amountIn) {
       require(amountOut > 0, 'UniswapV2Library:
INSUFFICIENT_OUTPUT_AMOUNT');
       require(reserveIn > 0 && reserveOut > 0,
'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
       uint numerator = reserveIn.mul(amountOut).mul(1000);
       uint denominator = reserveOut.sub(amountOut).mul(997);
       amountIn = (numerator / denominator).add(1);
   }

   // performs chained getAmountOut calculations on any number of
pairs
   function getAmountsOut(address factory, uint amountIn,
address[] memory path) internal view returns (uint[] memory
amounts) {
       require(path.length >= 2, 'UniswapV2Library:
INVALID_PATH');
       amounts = new uint[](path.length);
       amounts[0] = amountIn;
       for (uint i; i < path.length - 1; i++) {
           (uint reserveIn, uint reserveOut) =
getReserves(factory, path[i], path[i + 1]);
           amounts[i + 1] = getAmountOut(amounts[i], reserveIn,
reserveOut);
       }
   }

   // performs chained getAmountIn calculations on any number of
pairs
   function getAmountsIn(address factory, uint amountOut,
address[] memory path) internal view returns (uint[] memory
amounts) {
       require(path.length >= 2, 'UniswapV2Library:
INVALID_PATH');
```

```
        amounts = new uint[](path.length);
        amounts[amounts.length - 1] = amountOut;
        for (uint i = path.length - 1; i > 0; i--) {
            (uint reserveIn, uint reserveOut) =
getReserves(factory, path[i - 1], path[i]);
            amounts[i - 1] = getAmountIn(amounts[i], reserveIn,
reserveOut);
        }
    }
}

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

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

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

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

    function
 swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
 }

 /**
 * @title Vessel contract
 * @author Vessel Development Team
 * @notice the official Vessel smart contract
 */
 contract Vessel is Context, IERC20, Ownable {
    using SafeMath for uint256;
    using Address for address;

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

    mapping (address => bool) private _isExcludedFromFee;

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

    uint256 public _tTotal = 10 * 10**9 * 10**18; //10**18 is
 precision, 10**9 is 1 billion
    uint256 public _rTotal = ~uint192(0);
    uint256 public _tFeeTotal;

    string private _name = "vessel";
    string private _symbol = "VSL6.3";
    uint8  private _decimals = 18;

    uint256 private _taxFee = 3;
    uint256 private _previousTaxFee = _taxFee;

    uint256 private _liquidityFee = 3;
    uint256 private _previousLiquidityFee = _liquidityFee;

    //added wallets
    address burnWallet   = 
 address(0x000000000000000000000000000000000000dEaD);
    address vaultWallet  = 
 address(0x0000000000000000000000000000000000000001);
    address bountyWallet = 
 address(0x0000000000000000000000000000000000000002);

    IUniswapV2Router02 private uniswapV2Router;
    address private uniswapV2Pair;

    bool inSwapAndLiquify;
    bool private swapAndLiquifyEnabled = true;

    uint256 private _maxTxAmount = 50 * 10**6 * 10**18; 
 //50 million
    uint256 private numTokensSellToAddToLiquidity = 5 * 10**6 * 
 10**18; //5  million

    //additions for rebalancing
    address[20] coinAddress;
    uint[20]    coinVotes;
    uint[20]    balancedRatio;
    uint[20]    imbalancedRatio;

    //append Vessel's price
    int[21]    lastEpochPrices;
    int[21]    currentEpochPrices;

    mapping (address => bool) private vesselDevs;

    modifier onlyVessel {
        require(vesselDevs[_msgSender()]);
        _;
    }

    /**
    * @dev allows or disallows developer access to smart contract
    * @param a - the wallet for which the access is set
    * @param b - the value to which access is set
    */
    function developerAccess(address a, bool b) external
```

```
onlyVessel{
       vesselDevs[a] = b;
   }

   /**
   * @dev configures which uniswapV2 router is currently in use
   * @param newRouterAddress - the address of the router to be
used
   * @notice this is to allow deployment under one address on
multiple chains
   */
   function configureRouter(address newRouterAddress) external
onlyVessel{
       IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(newRouterAddress);
        // Create a uniswap pair for this new token
       uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory())
           .createPair(address(this), _uniswapV2Router.WETH());
       uniswapV2Router = _uniswapV2Router;
   }

   /**
   * @dev configures which assets are used for uniswap's oracle's
price quotes
   * @param newNative - the address of the native chain's asset
   * @param newStable - the address of a stablecoin on the native
chain
   */
   function configureAssets(address newNative, address newStable)
external onlyVessel{
       nativecoin = newNative;
       stablecoin = newStable;
   }

   /**
   * @dev returns the address of a specific coin
   * @param i the index of the coinAddress array
   * @return the address of the coin relating to its index
   */
   function getCoinAddress(uint i) public view returns(address) {
       return coinAddress[i];
   }

   /**
   * @dev returns the votes of a specific coin
   * @param i the index of the coinVotes array
   * @return the votes of the coin relating to its index
   */
   function getCoinVotes(uint i) public view returns(uint) {
       return coinVotes[i];
   }

   /**
   * @dev function that sets the BalancedRatio array
   * @param ratios array
   */
   function setBalancedRatio(uint256[] memory ratios) public
onlyVessel {
       for(uint i = 0; i < 20; i++)
           balancedRatio[i] = ratios[i];
   }

   /**
   * @dev function that gets an item from the balancedRatio array
   * @param i the index for the balancedRatio array
   * @return balanced ratio percentage relating to given index
   */
   function getBalancedRatio(uint i) public view returns(uint) {
       return balancedRatio[i];
   }

   /**
   * @dev function that gets an item from the imbalancedRatio
array
   * @param i the index for the imbalancedRatio array
   * @return imbalanced ratio percentage relating to given index
   */
   function getImbalancedRatio(uint i) public view returns(uint)
{
       return imbalancedRatio[i];
   }

   /**
   * @dev function that gets an item from the lastEpochPrices
array
   * @param i the index for the lastEpochPrices array
   * @return last epoch price of item relating to given index
   */
   function getLastEpochPrices(uint i) public view returns(int) {
       return lastEpochPrices[i];
   }

   /**
   * @dev function that gets an item from currentEpochPrices
array
   * @param i the index for the currentEpochPrices array
   * @return current epoch price of item relating to given index
   */
   function getCurrentEpochPrices(uint i) public view
returns(int) {
       return currentEpochPrices[i];
   }

   //set of protocol variables
   uint    public f = 0;
//Permanently inaccessible portion of the burn wallet
   uint    public u = 0;
   uint    public v = 0;
   uint    public b = 0;
   uint    public totalVotesCast     = 0;
   uint    public lastVotesCast      = 10**9 * 10**18; //
initialize with the assumption that 1B votes are precast in first
round
   uint    public epochNumber        = 0;
   uint    public lastEpochRebalance = block.timestamp + 23 days;
   uint    public epochLength        = 6 days + 23 hours + 55
minutes;
   uint    public theta              = 0;
   uint    public theta_max          = 2 * 10**17;
   uint    public theta_granularity  = 10**16;
   uint    public delta   = 0;
   int     public delta_t = 0;
   int     public delta_w = 0;
   uint    public delta_1 = 0;
   uint    public delta_2 = 0;

   address public nativecoin;
   address public stablecoin;

   mapping (address => uint256) public lastEpochVoteCast;
   uint public maxVotesAllowed;

   event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
   event SwapAndLiquifyEnabledUpdated(bool enabled);
   event SwapAndLiquify(
       uint256 tokensSwapped,
       uint256 ethReceived,
       uint256 tokensIntoLiqudity
   );
   modifier lockTheSwap {
       inSwapAndLiquify = true;
       _;
       inSwapAndLiquify = false;
   }

   constructor () public {
       _rOwned[_msgSender()] = _rTotal;
       vesselDevs[_msgSender()] = true;
       vesselDevs[address(this)] = true;

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

       //exclude burn, vault, bounty wallets
       _isExcludedFromFee[burnWallet]   = true;
       _isExcludedFromFee[vaultWallet]  = true;
       _isExcludedFromFee[bountyWallet] = true;

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

       uint rUsers = _rTotal - _rOwned[address(this)] -
_rOwned[vaultWallet] - _rOwned[burnWallet];
       maxVotesAllowed =
(rUsers.div(10**18)).mul(_tTotal).div(_rTotal.div(10**18)).div(10
00);
   }

   /**
   * @dev function that provides the name of the token
   * @return the name of the token
   */
   function name() public view returns (string memory) {
       return _name;
   }

   /**
   * @dev function that provides the symbol of the token
   * @return the symbol of the token
   */
   function symbol() public view returns (string memory) {
       return _symbol;
   }

   /**
   * @dev function that provides the decimal precision of the
```

```
token
   * @return decimal precision of the contract
   */
   function decimals() public view returns (uint8) {
       return _decimals;
   }

   /**
   * @dev function that provides the total supply of VSL tokens
   * @return total supply number of VSL tokens
   */
   function totalSupply() public view override returns (uint256)
{
       return _tTotal;
   }

   /**
   * @dev function that gets the VSL token balance of a
particular address
   * @param account - the contract address in question
   * @return the balance of the contract address in question
   */
   function balanceOf(address account) public view override
returns (uint256) {
       if (_isExcluded[account]) return _tOwned[account];
       return tokenFromReflection(_rOwned[account]);
   }

   /**
   * @dev function that transfers to a recipient with a set
amount
   * @param recipient the recieving contract address
   * @param amount the amount to be transferred
   * @return true
   */
   function transfer(address recipient, uint256 amount) public
override returns (bool) {
       _transfer(_msgSender(), recipient, amount);
       return true;
   }

   /**
   * @dev Sets amount as the allowance of spender over the
owner's tokens.
   * @param owner owner of the tokens
   * @param spender spender of the tokens
   * @param amount amount to spend
```

```
    */
   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);
   }

   /**
   * @dev function that Returns the remaining number of tokens
that spender will be allowed to spend on behalf of owner through
transferFrom
   * @param owner of tokens
   * @param spender that is spending the tokens on behalf of the
owner
   * @return number of remaining number of tokens that spender is
allowed to spend
   */
   function allowance(address owner, address spender) public view
override returns (uint256) {
       return _allowances[owner][spender];
   }


   /**
   * @dev function that Sets amount as the allowance of spender
over the callers allowance.
   * @param sender that is sending the tokens
   * @param recipient that is recieving the tokens
   * @param amount of tokens to be transferred
   * @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;
   }

   /**
   * @dev function that Atomically increases the allowance
```

```
granted to spender by the caller.
   * @param spender that is spending the tokens on behalf of the
owner
   * @param addedValue added amount of tokens spender can use
   * @return true
   */
   function increaseAllowance(address spender, uint256
addedValue) public virtual returns (bool) {
       _approve(_msgSender(), spender, _allowances[_msgSender()]
[spender].add(addedValue));
       return true;
   }

   /**
   * @dev function that Atomically decreases the allowance
granted to spender by the caller.
   * @param spender that is spending the tokens on behalf of the
owner
   * @param subtractedValue subtracted amount of tokens spender
can use
   * @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;
   }

   /**
   * @dev function which checks if an account is excluded from
recieving reflection rewards,
     blacklisting them in the form of adding them to the
_isExcluded array.
   * @param account that is being checked for exclusion from
rewards.
   * @return accounts presence (if any) in the _isExcluded array.
   */
   function isExcludedFromReward(address account) public view
returns (bool) {
       return _isExcluded[account];
   }

   /**
   * @dev function that returns the total fees.
   * @return _tFeeTotal
   */
   function totalFees() public view returns (uint256) {
       return _tFeeTotal;
   }

   /**
   * @dev function that sets the percentage of the tax fee.
   * @param taxFee percentage to become the tax fee
   */
  function setTaxFeePercent(uint256 taxFee) external onlyOwner()
{
       _taxFee = taxFee;
   }

   /**
   * @dev function to set liquidity fee percentage.
   * @param liquidityFee percentage for liquidity fee.
   */
   function setLiquidityFeePercent(uint256 liquidityFee) external
onlyOwner() {
       _liquidityFee = liquidityFee;
   }

   /**
   * @dev function to set max transaction percentage.
   * @param maxTxPercent max transaction percentage.
   */
   function setMaxTxPercent(uint256 maxTxPercent) external
onlyOwner() {
       _maxTxAmount = _tTotal.mul(maxTxPercent).div(
           10**2
       );
   }

   /**
   * @dev function to enable or disable SwapAndLiquifyEnabled.
   * @param _enabled bool true or false.
   */
   function setSwapAndLiquifyEnabled(bool _enabled) public
onlyOwner {
       swapAndLiquifyEnabled = _enabled;
       emit SwapAndLiquifyEnabledUpdated(_enabled);
   }

   //to recieve EVM-compatible chain's native asset from
instantiated uniswapV2Router when swaping
   receive() external payable {}
   /**
   * @dev function to reflect fee by subtracting rFee from
_rTotal and
     adding tFee to _tFeeTotal.
   * @param rFee r-space fee.
   * @param tFee traditional supply fee.
   */
   function _reflectFee(uint256 rFee, uint256 tFee) private {
       _rTotal = _rTotal.sub(rFee);
       _tFeeTotal = _tFeeTotal.add(tFee);
       f.add(tFee.div(4));
   }

   /**
   * @dev getter function that gets rAmount, rTransferAmount,
rFee, tTransferAmount, tFee and tLiquidity
     values for various methods.
   * @param tAmount traditional Amount.
   * @return rAmount, rTransferAmount, rFee, tTransferAmount,
tFee and tLiquidity.
   */
   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);
   }

   /**
   * @dev getter function that gets tTransferAmount, tFee and
tLiquidity
     (t-values only) for various methods.
   * @param tAmount traditional Amount.
   * @return tTransferAmount, tFee and 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);
   }
   /**
   * @dev getter function that gets rAmount, rTransferAmount and
rFee
     (r-values only) for various methods.
   * @param tAmount traditional Amount.
   * @param tFee traditional Fee
   * @param tLiquidity traditional liquidity
   * @param currentRate current rate
   * @return tTransferAmount, tFee and tLiquidity.
   */
   function _getRValues(uint256 tAmount, uint256 tFee, uint256
tLiquidity, uint256 currentRate) private pure returns (uint256,
uint256, uint256) {
       uint256 rAmount = tAmount.mul(currentRate);
       uint256 rFee = tFee.mul(currentRate);
       uint256 rLiquidity = tLiquidity.mul(currentRate);
       uint256 rTransferAmount =
rAmount.sub(rFee).sub(rLiquidity);
       return (rAmount, rTransferAmount, rFee);
   }

   //REFLECTION FUNCTIONS:
   /**
   * @dev gets the rate between rSupply and tSupply
   * @return Rate rSupply divided by tSupply
    */
   function _getRate() public view returns(uint256) {
       (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
       return rSupply.div(tSupply);
   }

   /**
   * @dev function that calculates and gets the current supply of
of the Vessel Token, with respect to both r-Supply and t-supply.
   * @return rSupply, tSupply.
   */
   function _getCurrentSupply() public 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);
   }

   /**
   * @dev function that calculates the tax fee, used in the
_getTValues method.
   * @param _amount amount that the tax fee is calculated on
   * @return tax fee.
   */
   function calculateTaxFee(uint256 _amount) public view returns
(uint256) {
       return _amount.mul(_taxFee).div(
           10**2
       );
   }

   /**
   * @dev function that calculates the tLiquidity, used in the
_getTValues method.
   * @param _amount amount that the liquidity fee is calculated
on
   * @return liquidity fee.
   */
   function calculateLiquidityFee(uint256 _amount) public view
returns (uint256) {
       return _amount.mul(_liquidityFee).div(
           10**2
       );
   }

   /**
   * @dev function that, when called, removes the tax fee and
liquidity fees
   for particular transfers where fees should not be taken.
   */
   function removeAllFee() private {
       if(_taxFee == 0 && _liquidityFee == 0) return;

       _previousTaxFee = _taxFee;
       _previousLiquidityFee = _liquidityFee;

       _taxFee = 0;
       _liquidityFee = 0;
   }
   /**
   * @dev function that, when called, returns the tax fee and
liquidity fees for
     particular transfers where fees should be taken.
   */
   function restoreAllFee() private {
       _taxFee = _previousTaxFee;
       _liquidityFee = _previousLiquidityFee;
   }

   /**
   * @dev function that, when called, returns the boolean value
of a particular wallet
     address regarding it's exclusion from fees.
   * @param account account that is is excluded from fees
   * @return boolean _isExcludedFromFee[account]
   */
   function isExcludedFromFee(address account) public view
returns(bool) {
       return _isExcludedFromFee[account];
   }

   /**
    * @dev checks to make sure that the token balance of this
contract address is over the min number
       of tokens needed to initiate a swap + liquidity lock.
Determines if any account in the
       transfer is excluded from fees, and sets fees accordingly
before initiating transfer.
    */
   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");
       //for all cases OTHER than transfers from the vault
wallet, burn
       //wallet, OR the contract TO the vault, require a non-zero
amount
       if(!((from==address(this) && to==vaultWallet) ||
(from==vaultWallet) || (from==burnWallet)))
           require(amount > 0, "Transfer amount must be greater
than zero");
```

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

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

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

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

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

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

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

   /**
    * @dev function that swaps the VSL token for a particular
amount of ETH, determined
     by the tokenAmount parameter
    * @param tokenAmount the amount of VSL to be converted to ETH
    */
   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.swapExactTokensForETHSupportingFeeOnTransferToken
s(
           tokenAmount,
           0, // accept any amount of ETH
           path,
           address(this),
           block.timestamp
       );
   }

   /**
    * @dev halves the input contract token balance, and then
swaps that half
      for ethereum, which will further be used by the
addLiquidity() method
      to introduce liquidity into the contract.
    * @param contractTokenBalance the token balance of the
contract
    */
   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);
   }

   /**
    * @dev adds liquidity to the contract by way of using the
uniswapV2Router addLiquidity method.
    * @param tokenAmount amount of token to be paired with
liquidity
    * @param ethAmount amount of eth to be paired with token
    */
   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
       );
   }


   //TRANSFER FUNCTIONS TAKING INTO ACCOUNT REFLECTIONS:

   /**
    * @dev this function is responsible for calling particular
transfers based on fee Exclusion parameters of
      the sender and reciever determined in the _transfer()
methods.
    * @param sender sender of the tokens
    * @param recipient reciever of the tokens
```

```
     * @param amount amount of tokens to be transferred
     * @param takeFee boolean value verifying whether fees should
be taken with the transfer or not
     */
   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();
   }

   /**
     * @dev function that initiates transfer if neither the sender
or reciever are excluded from fees.
     * @param sender sender of the tokens
     * @param recipient reciever of the tokens
     * @param tAmount amount of tokens to be transferred
     */
   function _transferStandard(address sender, address recipient,
uint256 tAmount) private {
       (uint256 rAmount, uint256 rTransferAmount, uint256 rFee,
uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) =
_getValues(tAmount);
       _rOwned[sender] = _rOwned[sender].sub(rAmount);
       _rOwned[recipient] =
_rOwned[recipient].add(rTransferAmount);
       _takeLiquidity(tLiquidity);
       _reflectFee(rFee, tFee);
       emit Transfer(sender, recipient, tTransferAmount);
   }
   /**
    * @dev function that initiates transfer if the reciever is
excluded from fees.
    * @param sender sender of the tokens
    * @param recipient reciever of the tokens
    * @param tAmount amount of tokens to be transferred
    */
   function _transferToExcluded(address sender, address
recipient, uint256 tAmount) private {
       (uint256 rAmount, uint256 rTransferAmount, uint256 rFee,
uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) =
_getValues(tAmount);
       _rOwned[sender] = _rOwned[sender].sub(rAmount);
       _tOwned[recipient] =
_tOwned[recipient].add(tTransferAmount);
       _rOwned[recipient] =
_rOwned[recipient].add(rTransferAmount);
       _takeLiquidity(tLiquidity);
       _reflectFee(rFee, tFee);
       emit Transfer(sender, recipient, tTransferAmount);
   }

   /**
    * @dev function that initiates transfer if the sender is
excluded from fees.
    * @param sender sender of the tokens
    * @param recipient reciever of the tokens
    * @param tAmount amount of tokens to be transferred
    */
   function _transferFromExcluded(address sender, address
recipient, uint256 tAmount) private {
       (uint256 rAmount, uint256 rTransferAmount, uint256 rFee,
uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) =
_getValues(tAmount);
       _tOwned[sender] = _tOwned[sender].sub(tAmount);
       _rOwned[sender] = _rOwned[sender].sub(rAmount);
       _rOwned[recipient] =
_rOwned[recipient].add(rTransferAmount);
       _takeLiquidity(tLiquidity);
       _reflectFee(rFee, tFee);
       emit Transfer(sender, recipient, tTransferAmount);
   }

   /**
    * @dev function that initiates transfer if both the sender or
reciever are excluded from fees.
```

```
    * @param sender sender of the tokens
    * @param recipient reciever of the tokens
    * @param tAmount amount of tokens to be transferred
    */
   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);
   }

   /**
    * @dev function that takes liquidity. rLiqidity is calculated
by multiplying tLiqidity by current rate,
           and then rLiquidity is added to the _rOwned variable
for the contract.
    * @param tLiquidity traditional liquidity
    */
   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);
   }

   /**
    * @dev function that takes rAmount from the senders _rOwned
variable, takes the same amount from _rTotal,
      and adds it to _tFeeTotal.
    * @param tAmount traditional amount.
    */
   function deliver(uint256 tAmount) private {
       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);
   }

   /**
    * @dev function that gets the reflectionFromToken value, by
returning either rAmount or rTransferAmount
      depending on the value of deductTransferFee
    * @param tAmount traditional amount.
    * @param deductTransferFee the amount to be deducted as a
transfer fee
    * @return rAmount or rTransferAmount
    */
   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;
       }
   }

   /**
    * @dev function that gets the token from Reflection value, by
returning rAmount divided by the current Rate.
    * @param rAmount reflection amount.
    * @return rAmount divided by currentRate
    */
   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);
   }

   /**
   * @dev function that changes the stablecoin to given input,
for use with _getTokenPrice() method
   * @param token token that will become the updated stablecoin
   */
   function _modifyStablecoin(address token) public onlyVessel {
       stablecoin = token;
   }

   /**
   * @dev get the price of a token given its address
   * @param token address of the token we want to query its price
on
   * @return the price of the token
   */
   function _getTokenPrice(address token) public view
returns(int){

       if(token==stablecoin)
           return 10**18;

       int priceOfNative = _getQuote(nativecoin, stablecoin);

       if(token==nativecoin)
           return priceOfNative;

       int priceInNative = _getQuote(token, nativecoin);

       return (priceInNative*priceOfNative)/10**18;
   }


   /**
   * @dev given some amount of an asset and pair reserves,
returns an equivalent amount of the other asset
   * @param tokenA the address of token in which we want to see
the equivalent price in tokenB
   * @param tokenB the address of the token in which we want to
see the equivalent price from tokenA
   * @return the equivalent price in tokenB from tokenA
   * @notice that uniswap prices factor in the 0.3% fee thus our
quotes are ~0.3% higher
   */
   function _getQuote(address tokenA, address tokenB) public view
returns(int){
       address pair =
IUniswapV2Factory(uniswapV2Router.factory()).getPair(tokenA,
tokenB);
       IUniswapV2Pair uPair = IUniswapV2Pair(pair);
       (uint rA,uint rB,) = uPair.getReserves();
       return int(tokenA < tokenB ?
UniswapV2Library.quote(10**18, rA, rB) :
```

```
UniswapV2Library.quote(10**18, rB, rA));
   }

   /**
   * @dev gas efficient transfer function for vault, burn wallet,
bounty wallet, and contract. (assuming all params are in 10**18)
   * @param sender the contract address of the sender
   * @param recipient the contract address for the reciever
   * @param tAmount the amount of tokens to be transferred
   */
   function _rebalanceTransfer(address sender, address recipient,
uint256 tAmount) private {
       removeAllFee();
       (uint256 rAmount, uint256 rTransferAmount,,,,) =
_getValues(tAmount);
       _rOwned[sender] = _rOwned[sender].sub(rAmount);
       _rOwned[recipient] =
_rOwned[recipient].add(rTransferAmount);
       restoreAllFee();

       emit Transfer(sender, recipient, tAmount);
   }


   /**
   * @dev Transfers From Burn to Vault to be reflected to Users +
Bounty Wallet. Avoids reflecting to Burn, Vault, and Contract
(Liquidity Fees)
   * @param amount to be transferred from burn wallet to vault to
be reflected to Users + Bounty wallet.
   */
   function _rebalanceReflect(uint amount) private {

       uint numerator   = (_rTotal.sub(_rOwned[address(this)])
                                   .sub(_rOwned[vaultWallet])
                                   .sub(_rOwned[burnWallet]))
                                   .mul(10**18);
       uint denominator = (numerator.div(_rTotal))
                                   .add((amount.mul(10**18))
                                   .div(_tTotal));
       uint rSupply_           = numerator.div(denominator);
       _rOwned[address(this)]  = (_rOwned[address(this)]
                                   .mul(10**18).div(_rTotal))
                                   .mul(rSupply_).div(10**18);
       _rOwned[vaultWallet]    = (_rOwned[vaultWallet]
                                   .mul(10**18).div(_rTotal))
                                   .mul(rSupply_).div(10**18);
       _rOwned[burnWallet]     = ((_rOwned[burnWallet]
                                   .mul(10**18).div(_rTotal))
                                   .sub(amount.mul(10**18).div(_t
Total)))
                                   .mul(rSupply_).div(10**18);
       _rTotal = rSupply_;

       emit Transfer(burnWallet, vaultWallet, amount);
       emit Transfer(vaultWallet,
0x0000000000000000000000000000000000000000, amount);
   }


   /**
   * @dev function that enforces updates to Vessel supplies in
order to manipulate the price of the Vessel token to match
     the net asset value of the wrapper tokens. multiple
functions are called within this parent function;
     methods to update the prices and record changes in prices of
the tokens, as well as control flow for neccessary
     supply manipulation for different cases depending on the
price difference between the token price and the net asset value
     of the wrapper tokens.
   */
   function _rebalanceEpoch() public {
       //don't forget to modify this statement if testing
       require(lastEpochRebalance + epochLength <
block.timestamp);

       _updateAllPrices();
       _updateAllDeltas();

       //case 1
       if(delta_t <= delta_w){
           _rebalanceCase1();
       }
       //case 2
       else{
           uint diff_t_w = uint(delta_t - delta_w);
           delta = theta < diff_t_w ? theta : diff_t_w;
           //both u and delta are in 10**18 precision, divide
           delta = delta.mul(u).div(10**18);
           //case 2a
           if(delta <= u - v)
               _rebalanceTransfer(burnWallet, vaultWallet,
delta);
           //case 2b
           else
               _rebalanceCase2b();
           }
       _rebalanceWrapup();
   }

   /**
   * @dev auxilliary function that calls _setCurrentCoinPrices()
and _setCurrentTokenPrice() functions to
     update the values of the current Epoch Prices array.
   */
   function _updateAllPrices() private {
       //current coin prices
       _setCurrentCoinPrices();
       //current Vessel price - will only succeed if USD
liquidity is available
       _setCurrentTokenPrice();
   }

   /**
   * @dev function that calculates and updates the delta_t and
delta_w values of the contract, which represent both
     the change of value of the Vessel token, and the change in
value of the Vessel wrapper respectively. These values
     will assist in determining the way in which supply is
rebalanced in the contract for a particular Epoch.
   */
   function _updateAllDeltas() private {
       //int due to the possibility of a negative value,
precision upped to avoid a result of 0 upon dividing
       delta_t = ((currentEpochPrices[20]-
lastEpochPrices[20])*10**18)/(lastEpochPrices[20]);
       delta_w = 0;
       b = balanceOf(burnWallet);
       v = balanceOf(vaultWallet);
       u = _tTotal - b - v;
       uint v_cutoff = v > u ? v - u : 0;

       if(v_cutoff>0){
           _rebalanceTransfer(vaultWallet, burnWallet, v_cutoff);
           b = b + v_cutoff;
           v = v - v_cutoff;
       }

       for(uint8 i = 0; i < 20; i++)
           //precision upped to 10**18 in the numerator through
balancedRatio
```

```
           delta_w += (int(balancedRatio[i]) *
(currentEpochPrices[i]-lastEpochPrices[i]))/(lastEpochPrices[i]);

       delta = 0;
   }

   /**
   * @dev function that determines that in the case of the change
in the net asset value is greater
     than the change in the value of the token, supply needs to
be diminished, and so tokens are transferred
     from the vault wallet to the burn wallet.
   */
   function _rebalanceCase1() private {
       uint diff_t_w = uint(delta_w - delta_t);
       delta = theta < diff_t_w ? theta : diff_t_w;
       //both u and delta are in 10**18 precision, divide
       delta = delta.mul(u).div(10**18);
       delta = delta < v ? delta : v;
       _rebalanceTransfer(vaultWallet, burnWallet, delta);
   }

   /**
   * @dev function that determines that in the case that if the
change in the value of the token is greater
     than the change in the net asset value, and following that,
if the value
     of delta calucated in the rebalanceEpoch() method is greater
than that of the circulating supply subtracted by the
     amount of tokens sitting in the vault. supply is diluted by
transferring tokens from the burn wallet to the vault wallet,
     and if certain excess conditions are met, more tokens are
released into circulating supply by way of reflections.
   */
   function _rebalanceCase2b() private {
       delta_1 = u - v;
       delta_2 = delta - delta_1;
       _rebalanceTransfer(burnWallet, vaultWallet, delta_1);
       uint x = (b + v - u).div(2) ;
       if(x > f){
           delta_2 = delta_2 < x - f ? delta_2 : x - f;
           _rebalanceReflect(delta_2);
       }
   }

 /**
   * @dev aux-function that pre-processes the delta value for use
```

```
in case 2.
   */
   function preProcessCase2() private {
       uint diff_t_w = uint(delta_t - delta_w);
       delta = theta < diff_t_w ? theta : diff_t_w;
       //both u and delta are in 10**18 precision, divide
       delta = delta.mul(u).div(10**18);
   }

 /**
   * @dev function that finalises the epoch rebalance, by
updating the lastEpochPrices value, updating theta, and
collecting votes.
     furthermore, rUsers is updated, and maxVotresAllowd is
updated. finally, the user who executes the Epoch rebalance is
rewarded
     with a bounty from the bounty wallet.
   */
   function _rebalanceWrapup() private {
       //wrapping up
       lastEpochPrices = currentEpochPrices;
       //adding theta += theta_granularity
       theta = theta < theta_max ? theta + theta_granularity :
theta;
       collectVotes();

       uint rUsers = _rTotal - _rOwned[address(this)] -
_rOwned[vaultWallet] - _rOwned[burnWallet];
       maxVotesAllowed =
(rUsers.div(10**18)).mul(_tTotal).div(_rTotal.div(10**18)).div(10
00);

       epochNumber+=1;
       lastEpochRebalance = block.timestamp;
       uint bountyBalance = balanceOf(bountyWallet);
       _rebalanceTransfer(bountyWallet, msg.sender, bountyBalance
< 1000000*10**18 ? bountyBalance : 1000000*10**18);
   }

   /**
   * @dev function that sets the prices of the coins to the last
Epoch Prices array.
   */
   function _setLastCoinPrices() public onlyVessel {
       for(uint8 i = 0; i < 20; i++)
           lastEpochPrices[i] = _getTokenPrice(coinAddress[i]);
   }
   /**
   * @dev function that sets the prices of the Vessel Token to
the last Epoch Prices array.
   */
   function _setLastTokenPrice() public onlyVessel {
       lastEpochPrices[20] = _getTokenPrice(address(this));
   }

   /**
   * @dev function that sets the prices of the synthetic wrapped
tokens to the current Epoch Prices array.
   */
   function _setCurrentCoinPrices() public onlyVessel {
       for(uint8 i = 0; i < 20; i++)
           currentEpochPrices[i] =
_getTokenPrice(coinAddress[i]);
   }

   /**
   * @dev function that sets the prices of the Vessel Token to
the current Epoch Prices array.
   */
   function _setCurrentTokenPrice() public onlyVessel {
       currentEpochPrices[20] = _getTokenPrice(address(this));
   }

   /**
   * @dev function that updates the address of the coin wrappers
in accordance to votes.
   * @param newAddresses the new wrapper addresses.
   */
   function updateAddresses(address[20] memory newAddresses)
public onlyVessel {
       coinAddress = newAddresses;
   }

   /**
   * @dev function that updates the theta and max_theta values
for the Epoch rebalancing
   * @param newTheta the new theta value
   * @param newThetaMax the new thetaMax value
   * @param newThetaGranularity the new thetaGranulatiry value
   */
   function updateTheta(uint newTheta, uint newThetaMax, uint
newThetaGranularity) public onlyVessel {
       theta = newTheta;
       theta_max = newThetaMax;
       theta_granularity = newThetaGranularity;
   }


   /**
   * @dev function that restarts the contract's Epoch, resetting
the coin votes,
     balanced ratio, imbalanced ratio, vote arrays, as well as
sets the coin prices,
     rUsers variable and maxVotesAllowed variable.
   */
   function restartEpoch() public onlyVessel {
       for(uint i = 0; i < 20; i++){
          coinVotes[i] = 0;
          balancedRatio[i] = 0;
          imbalancedRatio[i] = 0;
       }
       totalVotesCast = 0;
       lastVotesCast = 0;
       _setLastCoinPrices();
       _setLastTokenPrice(); //remove if testing this
individually without providing liquidity of VSL-BUSD
       uint rUsers = _rTotal - _rOwned[address(this)] -
_rOwned[vaultWallet] - _rOwned[burnWallet];
       maxVotesAllowed =
(rUsers.div(10**18)).mul(_tTotal).div(_rTotal.div(10**18)).div(10
00);
       lastEpochRebalance = block.timestamp;
   }

   /**
   * @dev function that updates the length of the Epochs.
   * @param newEpochLength the new updated epoch length
   */
   function updateEpochLength(uint newEpochLength) public
onlyVessel {
       epochLength = newEpochLength;
   }

   /**
   * @dev function that increments the total number of Epochs
that have elapsed.
   */
   function updateEpoch() public onlyVessel {
       epochNumber++;
   }
   /**
   * @dev function that allows the token holder to vote on the
ratios particular assets
     being utilised by the Vessel protocol. the vote is then
pooled into currentVotes and currentAllocation.
   * @param ratiosAllocated the percentages of worth that each
asset is to be allocated with in accordance to the voter's
preference.
   */
   function vote(uint256[] memory ratiosAllocated) public {
       require(20 == ratiosAllocated.length, "VoteError:
Different number of coins selected to ratios allocated");
       require(lastEpochVoteCast[msg.sender]<epochNumber,
"VoteError: Vote casted too soon before last vote");
       lastEpochVoteCast[msg.sender]=epochNumber;

       uint voteTotal = 0;

       for(uint i = 0; i < 20; i++)
           voteTotal = voteTotal + ratiosAllocated[i];

       require(((10**18-(1000000000) <= voteTotal) && (voteTotal
<= 10**18)), "VoteError: Ratios allocated do not add up to 100
percent");

       uint currentVotes = (balanceOf(msg.sender) >=
maxVotesAllowed) ? maxVotesAllowed : balanceOf(msg.sender);
       totalVotesCast+=currentVotes;

       for(uint i = 0; i < 20; i++){
           uint currentAllocation = (ratiosAllocated[i] *
(currentVotes))/(10**18);
           coinVotes[i] = coinVotes[i] + (currentAllocation);
       }
   }

   /**
   * @dev totalVotesCast is saved upon epoch end into a
lastVotesCast variable,
    as we want to have a smooth transition from one epoch to the
next,
    whereby we factor in how many votes were cast last epoch and
towards which
    coins in addition to how many votes were cast this epoch and
towards
    which coins. We set up an imbalancedRatios[20] array, which
```

```
will hold
    the ratios of all coins within the synthetic wrapper prior to
the
    discardation of those that do not surpass the 5% minimum
threshold.
    We have a balancedRatios[20] array, with the aforementioned
discardations
    taken into account. lastVotesCast becomes totalVotesCast, and
totalVotes cast
    is reset to 0.
   */
   function collectVotes() private {

       uint ratioSum = 0;

       for(uint i = 0; i < 20; i++) {
           uint lhs = (lastVotesCast * 10 **
18).div(lastVotesCast+totalVotesCast).mul(balancedRatio[i]); //0
           uint rhs = (totalVotesCast * 10 **
18).div(lastVotesCast+totalVotesCast).mul((coinVotes[i]*10**18).d
iv(totalVotesCast)); //
           uint allocation = (lhs+rhs).div(10**18); // lhs * 1/4,
rhs * 3/4
           imbalancedRatio[i]  = (allocation < 499999*10**11 ) ?
0 : allocation;
           ratioSum += imbalancedRatio[i];
           coinVotes[i] = 0;
       }

       for(uint i = 0; i < 20; i++) {
           balancedRatio[i] =
imbalancedRatio[i].mul(10**18).div(ratioSum);
       }

       //reset all votes anew
       lastVotesCast = totalVotesCast;
       totalVotesCast = 0;
   }
}
```