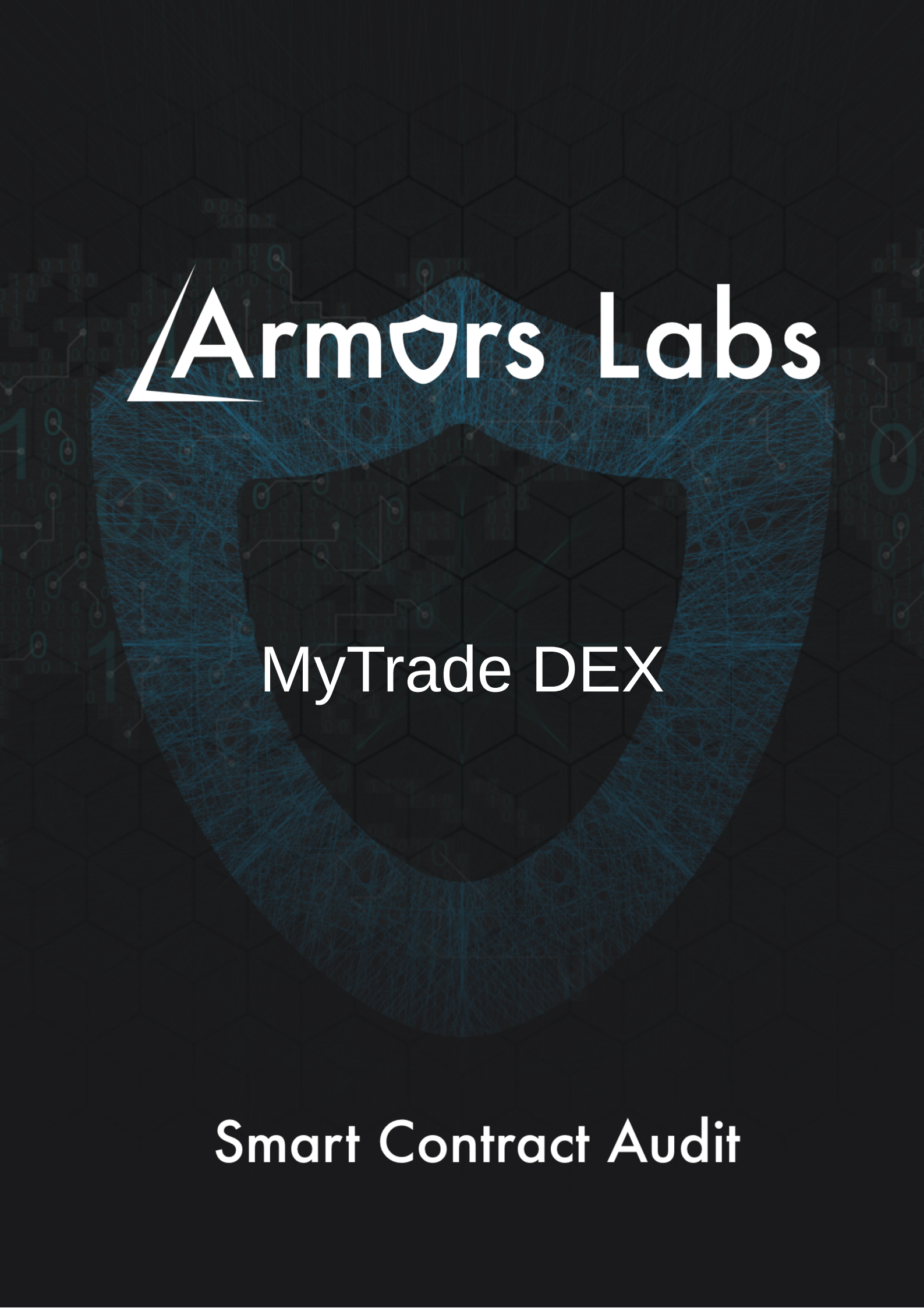

Armors Labs
MyTrade DEX
Smart Contract Audit

MyTrade DEX Audit Summary

Project name : MyTrade DEX Contract

Project address: None

Code URL : https://github.com/mytrade-dex/mytrade

Commit : 26c0c546dae4f9355d4304b61b8e3bbc1a9848ef

Project target : MyTrade DEX Contract Audit

Blockchain : Polygon

Test result : PASSED

Audit Info

Audit NO : 0X202109170006

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

MyTrade DEX Audit

The MyTrade DEX team asked us to review and audit their MyTrade DEX contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
MyTrade DEX Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-09-17

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the MyTrade DEX contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
./MytradeSwapMining.sol	c5085344617623ce53e522d2325aa8cf
./MyTradeOrderBook.sol	7a156f0b2d67557ffc08598f106be99d
./MyTradeOrderMining.sol	180714c9295ef56c34f50cdceed183ed
./MytradeRouter.sol	6a0163c27879bc32a9ef241177688b59
./MyTradeOrderBookExt.sol	c219299f792389b2652bd9d80b4bf930
./MyTradePool.sol	ffd4cce4dc42cd91135ec196af9a2dde
./MytradeFactory.sol	9584c29a88d0260665b5d4b65c6e6afa

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe

Vulnerability	status
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file


```
pragma solidity =0.5.16;

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

    function feeTo() external view returns (address);

    function feeToSetter() external view returns (address);

    function feeToRate() external view returns (uint256);

    function initCodeHash() external view returns (bytes32);

    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;

    function setFeeToRate(uint256) external;

    function setInitCodeHash(bytes32) external;

    function sortTokens(address tokenA, address tokenB) external pure returns (address token0, addres

    function pairFor(address tokenA, address tokenB) external view returns (address pair);

    function getReserves(address tokenA, address tokenB) external view returns (uint256 reserveA, uin
```

```
    function quote(uint256 amountA, uint256 reserveA, uint256 reserveB) external pure returns (uint25

    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) external view retu

    function getAmountIn(uint256 amountOut, uint256 reserveIn, uint256 reserveOut) external view retu

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

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

interface IMytradePair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);
    function setOrderbook(address _orderbook)external;
    function setRouter(address _router) external;
    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, by

    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 blockTim

    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 price(address token, uint256 baseDecimal) external view returns (uint256);

    function initialize(address, address) external;
}

interface IMytradeERC20 {
    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, by
}

interface IERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external view returns (string memory);

    function symbol() external view returns (string memory);

    function decimals() external view 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);
}

interface IHswapV2Callee {
    function hswapV2Call(address sender, uint amount0, uint amount1, bytes calldata data) external;
}

contract MytradeERC20 is IMytradeERC20 {
    using SafeMath for uint;

    string public constant name = 'MyTrade Swap LP Token';
    string public constant symbol = 'MyTradeLp';
    uint8 public constant decimals = 18;
    uint  public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

    bytes32 public DOMAIN_SEPARATOR;
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)"
    bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c6
    mapping(address => uint) public nonces;

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

    constructor() public {
        uint chainId;
        assembly {
            chainId := chainid
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingC
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }

    function _mint(address to, uint value) internal {
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
    }

    function _burn(address from, uint value) internal {
        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(address owner, address spender, uint value) private {
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(address from, address to, uint value) private {
        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
```

```
        emit Transfer(from, to, value);
    }

    function approve(address spender, uint value) external returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    function transfer(address to, uint value) external returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    function transferFrom(address from, address to, uint value) external returns (bool) {
        if (allowance[from][msg.sender] != uint(- 1)) {
            allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
        }
        _transfer(from, to, value);
        return true;
    }

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by
        require(deadline >= block.timestamp, 'MytradeSwap: EXPIRED');
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadlin
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'MytradeSwap: INVALID_SI
        _approve(owner, spender, value);
    }
}

contract MytradePair is IMytradePair, MytradeERC20 {
    using SafeMath  for uint;
    using UQ112x112 for uint224;

    uint public constant MINIMUM_LIQUIDITY = 10 ** 3;
    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));

    address public factory;
    address public token0;
    address public token1;

    uint112 private reserve0;           // uses single storage slot, accessible via getReserves
    uint112 private reserve1;           // uses single storage slot, accessible via getReserves
    uint32  private blockTimestampLast; // uses single storage slot, accessible via getReserves

    uint public price0CumulativeLast;
    uint public price1CumulativeLast;
    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity even

    // orderbook address
    address public orderbook;
    modifier onlyOrderbookOrRouter() {
        require(msg.sender == orderbook||msg.sender == router, "MytradePair: caller is not the orderb
        _;
    }
    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'MytradeSwap: LOCKED');
        unlocked = 0;
        _;
```

```
        unlocked = 1;
    }
    // router address
    address public router;
    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTi
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }
    function setOrderbook(address _orderbook) external {
        if(orderbook==address(0)){
            orderbook = _orderbook;
        }
    }
    function setRouter(address _router) external {
        if(router==address(0)){
            router = _router;
        }
    }
    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'MytradeSwap: TRANSFER_FAI
    }

    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);
    address public flashLoan;
    constructor() public {
        factory = msg.sender;
         flashLoan=address(new FlashLoan(IMytradeFactory(factory).feeTo()));
    }

    function safeApproveFlashLoan(
        address tokenA
    )public{
        TransferHelper.safeApprove(
            tokenA,
            flashLoan,
            ~uint256(0)
        );
    }
    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'MytradeSwap: FORBIDDEN');
        // sufficient check
        token0 = _token0;
        token1 = _token1;
    }

    // update reserves and, on the first call per block, price accumulators
    function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1) private {
        require(balance0 <= uint112(- 1) && balance1 <= uint112(- 1), 'MytradeSwap: OVERFLOW');
        uint32 blockTimestamp = uint32(block.timestamp % 2 ** 32);
        uint32 timeElapsed = blockTimestamp - blockTimestampLast;
        // overflow is desired
        if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
            // * never overflows, and + overflow is desired
```

```
            price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed;
            price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;
        }
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        blockTimestampLast = blockTimestamp;
        emit Sync(reserve0, reserve1);
    }

    // if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
    function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool feeOn) {
        address feeTo = IMytradeFactory(factory).feeTo();
        feeOn = feeTo != address(0);
        uint _kLast = kLast;
        // gas savings
        if (feeOn) {
            if (_kLast != 0) {
                uint rootK = SafeMath.sqrt(uint(_reserve0).mul(_reserve1));
                uint rootKLast = SafeMath.sqrt(_kLast);
                if (rootK > rootKLast) {
                    uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                    uint denominator = rootK.mul(IMytradeFactory(factory).feeToRate()).add(rootKLast)
                    uint liquidity = numerator / denominator;
                    if (liquidity > 0) _mint(feeTo, liquidity);
                }
            }
        } else if (_kLast != 0) {
            kLast = 0;
        }
    }

    // this low-level function should be called from a contract which performs important safety check
    function mint(address to) external lock returns (uint liquidity) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        // gas savings
        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0.sub(_reserve0);
        uint amount1 = balance1.sub(_reserve1);

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply;
        // gas savings, must be defined here since totalSupply can update in _mintFee
        if (_totalSupply == 0) {
            liquidity = SafeMath.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
            _mint(address(0), MINIMUM_LIQUIDITY);
            // permanently lock the first MINIMUM_LIQUIDITY tokens
        } else {
            liquidity = SafeMath.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply)
        }
        require(liquidity > 0, 'MytradeSwap: INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1);
        // reserve0 and reserve1 are up-to-date
        emit Mint(msg.sender, amount0, amount1);
    }

    // this low-level function should be called from a contract which performs important safety check
    function burn(address to) external lock returns (uint amount0, uint amount1) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        // gas savings
        address _token0 = token0;
        // gas savings
        address _token1 = token1;
```

```
        // gas savings
        uint balance0 = IERC20(_token0).balanceOf(address(this));
        uint balance1 = IERC20(_token1).balanceOf(address(this));
        uint liquidity = balanceOf[address(this)];

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply;
        // gas savings, must be defined here since totalSupply can update in _mintFee
        amount0 = liquidity.mul(balance0) / _totalSupply;
        // using balances ensures pro-rata distribution
        amount1 = liquidity.mul(balance1) / _totalSupply;
        // using balances ensures pro-rata distribution
        require(amount0 > 0 && amount1 > 0, 'MytradeSwap: INSUFFICIENT_LIQUIDITY_BURNED');
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1);
        // reserve0 and reserve1 are up-to-date
        emit Burn(msg.sender, amount0, amount1, to);
    }

    // this low-level function should be called from a contract which performs important safety check
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external onlyOrd
        require(amount0Out > 0 || amount1Out > 0, 'MytradeSwap: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'MytradeSwap: INSUFFICIENT_LIQUIDIT

        uint balance0;
        uint balance1;
        {// scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'MytradeSwap: INVALID_TO');
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out);
            // optimistically transfer tokens
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out);
            // optimistically transfer tokens
            if (data.length > 0) IHswapV2Callee(to).hswapV2Call(msg.sender, amount0Out, amount1Out, d
            balance0 = IERC20(_token0).balanceOf(address(this));
            balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'MytradeSwap: INSUFFICIENT_INPUT_AMOUNT');
        {// scope for reserve{0,1}Adjusted, avoids stack too deep errors
            uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
            uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
            require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }

    // force balances to match reserves
    function skim(address to) external lock {
        address _token0 = token0;
        // gas savings
        address _token1 = token1;
        // gas savings
        _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
```

```
        _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }

    // force reserves to match balances
    function sync() external lock {
        _update(IERC20(token0).balanceOf(address(this)), IERC20(token1).balanceOf(address(this)), res
    }

    function price(address token, uint256 baseDecimal) public view returns (uint256) {
        if ((token0 != token && token1 != token) || 0 == reserve0 || 0 == reserve1) {
            return 0;
        }
        if (token0 == token) {
            return uint256(reserve1).mul(baseDecimal).div(uint256(reserve0));
        } else {
            return uint256(reserve0).mul(baseDecimal).div(uint256(reserve1));
        }
    }

}

contract MytradeFactory is IMytradeFactory {
    using SafeMath for uint256;
    address public feeTo;
    address public feeToSetter;
    uint256 public feeToRate;
    bytes32 public initCodeHash;
    bool public initCode = false;
    // orderbook address
    address public orderbook;
    address public router;
    mapping(address => mapping(address => address)) public getPair;
    address[] public allPairs;

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

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
    }

    function setOrderbook(address _orderbook) external {
        if(orderbook==address(0)){
            orderbook = _orderbook;
        }
    }
    function setRouter(address _router) external {
        if(router==address(0)){
            router = _router;
        }
    }
    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'MytradeSwapFactory: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'MytradeSwapFactory: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'MytradeSwapFactory: PAIR_EXISTS');
        // single check is sufficient
        bytes memory bytecode = type(MytradePair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IMytradePair(pair).initialize(token0, token1);
```

```
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair;
        // populate mapping in the reverse direction
        allPairs.push(pair);
        IMytradePair(pair).setOrderbook(orderbook);
        IMytradePair(pair).setRouter(router);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }
// calculates the CREATE2 address for a pair without making any external calls
    function pairFor(address tokenA, address tokenB) public view returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(uint(keccak256(abi.encodePacked(
                hex'ff',
                address(this),
                keccak256(abi.encodePacked(token0, token1)),
                initCodeHash
            ))));
    }
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'MytradeSwapFactory: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'MytradeSwapFactory: FORBIDDEN');
        require(_feeToSetter != address(0), "MytradeSwapFactory: FeeToSetter is zero address");
        feeToSetter = _feeToSetter;
    }

    function setFeeToRate(uint256 _rate) external {
        require(msg.sender == feeToSetter, 'MytradeSwapFactory: FORBIDDEN');
        require(_rate > 0, "MytradeSwapFactory: FEE_TO_RATE_OVERFLOW");
        feeToRate = _rate.sub(1);
    }

    function setInitCodeHash(bytes32 _initCodeHash) external {
        require(msg.sender == feeToSetter, 'MytradeSwapFactory: FORBIDDEN');
        require(initCode == false, "MytradeSwapFactory: Do not repeat settings initCodeHash");
        initCodeHash = _initCodeHash;
        initCode = true;
    }

    // returns sorted token addresses, used to handle return values from pairs sorted in this order
    function sortTokens(address tokenA, address tokenB) public pure returns (address token0, address
        require(tokenA != tokenB, 'MytradeSwapFactory: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'MytradeSwapFactory: ZERO_ADDRESS');
    }


    function getInitCodeHash() public pure returns (bytes32) {
        return keccak256(abi.encodePacked(type(MytradePair).creationCode));
    }


    // fetches and sorts the reserves for a pair
    function getReserves(address tokenA, address tokenB) public view returns (uint reserveA, uint res
        (address token0,) = sortTokens(tokenA, tokenB);
        (uint reserve0, uint reserve1,) = IMytradePair(pairFor(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 ass
    function quote(uint amountA, uint reserveA, uint reserveB) public pure returns (uint amountB) {
        require(amountA > 0, 'MytradeSwapFactory: INSUFFICIENT_AMOUNT');
        require(reserveA > 0 && reserveB > 0, 'MytradeSwapFactory: INSUFFICIENT_LIQUIDITY');
```

```
        amountB = amountA.mul(reserveB) / reserveA;
    }

    // given an input amount of an asset and pair reserves, returns the maximum output amount of the
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) public view returns (uint a
        require(amountIn > 0, 'MytradeSwapFactory: INSUFFICIENT_INPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'MytradeSwapFactory: 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 o
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) public view returns (uint a
        require(amountOut > 0, 'MytradeSwapFactory: INSUFFICIENT_OUTPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'MytradeSwapFactory: 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(uint amountIn, address[] memory path) public view returns (uint[] memory a
        require(path.length >= 2, 'MytradeSwapFactory: INVALID_PATH');
        amounts = new uint[](path.length);
        amounts[0] = amountIn;
        for (uint i; i < path.length - 1; i++) {
            (uint reserveIn, uint reserveOut) = getReserves(path[i], path[i + 1]);
            amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
        }
    }

    // performs chained getAmountIn calculations on any number of pairs
    function getAmountsIn(uint amountOut, address[] memory path) public view returns (uint[] memory a
        require(path.length >= 2, 'MytradeSwapFactory: 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(path[i - 1], path[i]);
            amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut);
        }
    }
}

library SafeMath {
    uint256 constant WAD = 10 ** 18;
    uint256 constant RAY = 10 ** 27;

    function wad() public pure returns (uint256) {
        return WAD;
    }

    function ray() public pure returns (uint256) {
        return RAY;
    }

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

        return c;
    }

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

```
    }

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

        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // 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;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        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;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }

    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a <= b ? a : b;
    }

    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    function sqrt(uint256 a) internal pure returns (uint256 b) {
        if (a > 3) {
            b = a;
            uint256 x = a / 2 + 1;
            while (x < b) {
                b = x;
                x = (a / x + x) / 2;
            }
        } else if (a != 0) {
            b = 1;
        }
    }
```

```
    function wmul(uint256 a, uint256 b) internal pure returns (uint256) {
        return mul(a, b) / WAD;
    }

    function wmulRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, b), WAD / 2) / WAD;
    }

    function rmul(uint256 a, uint256 b) internal pure returns (uint256) {
        return mul(a, b) / RAY;
    }

    function rmulRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, b), RAY / 2) / RAY;
    }

    function wdiv(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(mul(a, WAD), b);
    }

    function wdivRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, WAD), b / 2) / b;
    }

    function rdiv(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(mul(a, RAY), b);
    }

    function rdivRound(uint256 a, uint256 b) internal pure returns (uint256) {
        return add(mul(a, RAY), b / 2) / b;
    }

    function wpow(uint256 x, uint256 n) internal pure returns (uint256) {
        uint256 result = WAD;
        while (n > 0) {
            if (n % 2 != 0) {
                result = wmul(result, x);
            }
            x = wmul(x, x);
            n /= 2;
        }
        return result;
    }

    function rpow(uint256 x, uint256 n) internal pure returns (uint256) {
        uint256 result = RAY;
        while (n > 0) {
            if (n % 2 != 0) {
                result = rmul(result, x);
            }
            x = rmul(x, x);
            n /= 2;
        }
        return result;
    }
}

// a library for handling binary fixed point numbers (https://en.wikipedia.org/wiki/Q_(number_format)

// range: [0, 2**112 - 1]
// resolution: 1 / 2**112

library UQ112x112 {
    uint224 constant Q112 = 2 ** 112;

    // encode a uint112 as a UQ112x112
```

```
    function encode(uint112 y) internal pure returns (uint224 z) {
        z = uint224(y) * Q112;
        // never overflows
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function uqdiv(uint224 x, uint112 y) internal pure returns (uint224 z) {
        z = x / uint224(y);
    }
}
interface IFlashLoanService {
     function check(address from, uint256 value) external returns (bool);
}
/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
    address private _owner;

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

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor () public{
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     * @notice Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
```

```
    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0));
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
contract ReentrancyGuard {
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;
    uint256 private _status;
    constructor () public{
        _status = _NOT_ENTERED;
    }
    modifier nonReentrant() {
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
        _status = _ENTERED;
        _;
        _status = _NOT_ENTERED;
    }
}
// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MytradeSw
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MytradeSw
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MytradeSw
    }

}
contract FlashLoan is ReentrancyGuard,Ownable{
    address public loanServiceAddr;
    constructor(
        address _owner
    )public{
       _transferOwnership(_owner);
    }
    function setLoanServiceAddr(address _loanServiceAddr)public onlyOwner {
        loanServiceAddr=_loanServiceAddr;
    }
    function loan(
        address from,
        address token,
```

```
        uint value,
        address contractAddr,
        bytes memory msgdata
    ) public nonReentrant returns(bool){
       uint fromBal=IERC20(token).balanceOf(from);
       require(fromBal>=value,"insufficient balance");
       TransferHelper.safeTransferFrom(token,from,contractAddr,value);
       (bool success, bytes memory data) =contractAddr.call(msgdata);
       require(success && (data.length == 0 || abi.decode(data, (bool))), 'loan failed');
       require(IFlashLoanService(loanServiceAddr).check(from,fromBal));
       require(IERC20(token).balanceOf(from)>=fromBal,"error balance:loan failed");
       return true;
    }
}// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

contract Ownable {
    address private _owner;

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

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor () public{
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     * @notice Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
```

```
     * @param newOwner The address to transfer ownership to.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0));
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
contract ReentrancyGuard {
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;
    uint256 private _status;
    constructor () public{
        _status = _NOT_ENTERED;
    }
    modifier nonReentrant() {
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
        _status = _ENTERED;
        _;
        _status = _NOT_ENTERED;
    }
}
library SafeMath {
    /**
     * @dev Multiplies two unsigned integers, reverts on 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-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
     * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;
```

```
        return c;
    }

    /**
     * @dev Adds two unsigned integers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}

/**
 * @title ERC20 interface
 * @dev see https://eips.ethereum.org/EIPS/eip-20
 */
interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns (uint256);

}
interface IUniswapV2Factory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}
interface IUniswapV2Pair {
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
    function token0() external view returns (address);
    function token1() external view returns (address);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
}
// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
```

```
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'TransferHelper: MyTradeOrderBook ETH_TRANSFER_FAILED');
    }
}
/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 * ```
 * contract Example {
 *     // Add the library methods
 *     using EnumerableSet for EnumerableSet.AddressSet;
 *
 *     // Declare a set state variable
 *     EnumerableSet.AddressSet private mySet;
 * }
 * ```
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 */
library EnumerableSet {
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;

        // Position of the value in the `values` array, plus 1 because index 0
        // means a value is not in the set.
        mapping (bytes32 => uint256) _indexes;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function _add(Set storage set, bytes32 value) private returns (bool) {
        if (!_contains(set, value)) {
            set._values.push(value);
            // The value is stored at length-1, but we add 1 to all indexes
            // and use 0 as a sentinel value
            set._indexes[value] = set._values.length;
            return true;
```

```
        } else {
            return false;
        }
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) { // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the element to delete wit
            // the array, and then remove the last element (sometimes called as 'swap and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            // When the value to delete is the last one, the swap operation is unnecessary. However,
            // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' stateme

            bytes32 lastvalue = set._values[lastIndex];

            // Move the last value to the index where the value to delete is
            set._values[toDeleteIndex] = lastvalue;
            // Update the index for the moved value
            set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];

            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
```

```
     *
     * - `index` must be strictly less than {length}.
     */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        require(set._values.length > index, "EnumerableSet: index out of bounds");
        return set._values[index];
    }

    // Bytes32Set

    struct Bytes32Set {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _add(set._inner, value);
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _remove(set._inner, value);
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) {
        return _contains(set._inner, value);
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
        return _at(set._inner, index);
    }

    // AddressSet

    struct AddressSet {
        Set _inner;
```

```
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(AddressSet storage set, uint256 index) internal view returns (address) {
        return address(uint160(uint256(_at(set._inner, index))));
    }


    // UintSet

    struct UintSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }
```

```
    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
}
interface IMasterChefHeco {
    function pending(uint256 pid, address user) external view returns (uint256);

    function deposit(uint256 pid, uint256 amount) external;

    function withdraw(uint256 pid, uint256 amount) external;

    function emergencyWithdraw(uint256 pid) external;
}
contract MyTradePool is Ownable {
    using SafeMath for uint256;

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _multLP;

    // Info of each user.
    struct UserInfo {
        uint256 amount;     // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt.
        uint256 multLpRewardDebt; //multLp Reward debt.
    }

    // Info of each pool.
    struct PoolInfo {
        IERC20 lpToken;           // Address of LP token contract.
        uint256 allocPoint;       // How many allocation points assigned to this pool. tokens to dist
        uint256 lastRewardBlock;  // Last block number that tokens distribution occurs.
```

```
        uint256 accTokenPerShare; // Accumulated Tokens per share, times 1e12.
        uint256 accMultLpPerShare; //Accumulated multLp per share
        uint256 totalAmount;    // Total amount of current pool deposit.
    }

    // The IERC20 Token!
    IERC20 public myt;
    // Token tokens created per block.
    uint256 public tokenPerBlock;
    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;
    // Corresponding to the pid of the multLP pool
    mapping(uint256 => uint256) public poolCorrespond;
    // pid corresponding address
    mapping(address => uint256) public LpOfPid;
    // Control mining
    bool public paused = false;
    // Total allocation points. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when token mining starts.
    uint256 public startBlock;
    // multLP MasterChef
    address public multLpChef;
    // multLP Token
    address public multLpToken;
    // How many blocks are halved
    uint256 public halvingPeriod = 5256000;

    event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
    event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

    constructor(
        IERC20 _token,
        uint256 _tokenPerBlock,
        uint256 _startBlock
    ) public {
        myt = _token;
        tokenPerBlock = _tokenPerBlock;
        startBlock = _startBlock;
    }

    function setHalvingPeriod(uint256 _block) public onlyOwner {
        halvingPeriod = _block;
    }

    // Set the number of token produced by each block
    function setTokenPerBlock(uint256 _newPerBlock) public onlyOwner {
        massUpdatePools();
        tokenPerBlock = _newPerBlock;
    }

    function poolLength() public view returns (uint256) {
        return poolInfo.length;
    }

    function addMultLP(address _addLP) public onlyOwner returns (bool) {
        require(_addLP != address(0), "LP is the zero address");
        IERC20(_addLP).approve(multLpChef, uint256(- 1));
        return EnumerableSet.add(_multLP, _addLP);
    }

    function isMultLP(address _LP) public view returns (bool) {
        return EnumerableSet.contains(_multLP, _LP);
```

```
    }

    function getMultLPLength() public view returns (uint256) {
        return EnumerableSet.length(_multLP);
    }

    function getMultLPAddress(uint256 _pid) public view returns (address){
        require(_pid <= getMultLPLength() - 1, "not find this multLP");
        return EnumerableSet.at(_multLP, _pid);
    }

    function setPause() public onlyOwner {
        paused = !paused;
    }

    function setMultLP(address _multLpToken, address _multLpChef) public onlyOwner {
        require(_multLpToken != address(0) && _multLpChef != address(0), "is the zero address");
        multLpToken = _multLpToken;
        multLpChef = _multLpChef;
    }

    function replaceMultLP(address _multLpToken, address _multLpChef) public onlyOwner {
        require(_multLpToken != address(0) && _multLpChef != address(0), "is the zero address");
        require(paused == true, "No mining suspension");
        multLpToken = _multLpToken;
        multLpChef = _multLpChef;
        uint256 length = getMultLPLength();
        while (length > 0) {
            address dAddress = EnumerableSet.at(_multLP, 0);
            uint256 pid = LpOfPid[dAddress];
            IMasterChefHeco(multLpChef).emergencyWithdraw(poolCorrespond[pid]);
            EnumerableSet.remove(_multLP, dAddress);
            length--;
        }
    }

    // Add a new lp to the pool. Can only be called by the owner.
    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
        require(address(_lpToken) != address(0), "_lpToken is the zero address");
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
        lpToken : _lpToken,
        allocPoint : _allocPoint,
        lastRewardBlock : lastRewardBlock,
        accTokenPerShare : 0,
        accMultLpPerShare : 0,
        totalAmount : 0
        }));
        LpOfPid[address(_lpToken)] = poolLength() - 1;
    }

    // Update the given pool's Token allocation point. Can only be called by the owner.
    function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // The current pool corresponds to the pid of the multLP pool
```

```
    function setPoolCorr(uint256 _pid, uint256 _sid) public onlyOwner {
        require(_pid <= poolLength() - 1, "not find this pool");
        poolCorrespond[_pid] = _sid;
    }

    function phase(uint256 blockNumber) public view returns (uint256) {
        if (halvingPeriod == 0) {
            return 0;
        }
        if (blockNumber > startBlock) {
            return (blockNumber.sub(startBlock).sub(1)).div(halvingPeriod);
        }
        return 0;
    }

    function reward(uint256 blockNumber) public view returns (uint256) {
        uint256 _phase = phase(blockNumber);
        return tokenPerBlock.div(2 ** _phase);
    }

    function getTokenBlockReward(uint256 _lastRewardBlock) public view returns (uint256) {
        uint256 blockReward = 0;
        uint256 n = phase(_lastRewardBlock);
        uint256 m = phase(block.number);
        while (n < m) {
            n++;
            uint256 r = n.mul(halvingPeriod).add(startBlock);
            blockReward = blockReward.add((r.sub(_lastRewardBlock)).mul(reward(r)));
            _lastRewardBlock = r;
        }
        blockReward = blockReward.add((block.number.sub(_lastRewardBlock)).mul(reward(block.number)))
        return blockReward;
    }

    // Update reward variables for all pools. Be careful of gas spending!
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply;
        if (isMultLP(address(pool.lpToken))) {
            if (pool.totalAmount == 0) {
                pool.lastRewardBlock = block.number;
                return;
            }
            lpSupply = pool.totalAmount;
        } else {
            lpSupply = pool.lpToken.balanceOf(address(this));
            if (lpSupply == 0) {
                pool.lastRewardBlock = block.number;
                return;
            }
        }
        uint256 blockReward = getTokenBlockReward(pool.lastRewardBlock);
        if (blockReward <= 0) {
            return;
        }
```

```
        uint256 tokenReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
        TransferHelper.safeTransferFrom(
                address(myt),
                totalPoolAddr,
                address(this),
                tokenReward
            );

        pool.accTokenPerShare = pool.accTokenPerShare.add(tokenReward.mul(1e12).div(lpSupply));
        pool.lastRewardBlock = block.number;
    }
    address public totalPoolAddr;
    function setTotalPoolAddr(
        address _totalPoolAddr
    ) onlyOwner public returns(bool) {
        totalPoolAddr=_totalPoolAddr;
        return true;
    }
    // View function to see pending MDXs on frontend.
    function pending(uint256 _pid, address _user) external view returns (uint256, uint256){
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            (uint256 mytAmount, uint256 tokenAmount) = pendingMytAndToken(_pid, _user);
            return (mytAmount, tokenAmount);
        } else {
            uint256 mytAmount = pendingMyt(_pid, _user);
            return (mytAmount, 0);
        }
    }

    function pendingMytAndToken(uint256 _pid, address _user) private view returns (uint256, uint256){
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accTokenPerShare = pool.accTokenPerShare;
        uint256 accMultLpPerShare = pool.accMultLpPerShare;
        if (user.amount > 0) {
            uint256 TokenPending = IMasterChefHeco(multLpChef).pending(poolCorrespond[_pid], address(
            accMultLpPerShare = accMultLpPerShare.add(TokenPending.mul(1e12).div(pool.totalAmount));
            uint256 userPending = user.amount.mul(accMultLpPerShare).div(1e12).sub(user.multLpRewardD
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getTokenBlockReward(pool.lastRewardBlock);
                uint256 tokenReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accTokenPerShare = accTokenPerShare.add(tokenReward.mul(1e12).div(pool.totalAmount));
                return (user.amount.mul(accTokenPerShare).div(1e12).sub(user.rewardDebt), userPending
            }
            if (block.number == pool.lastRewardBlock) {
                return (user.amount.mul(accTokenPerShare).div(1e12).sub(user.rewardDebt), userPending
            }
        }
        return (0, 0);
    }

    function pendingMyt(uint256 _pid, address _user) private view returns (uint256){
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accTokenPerShare = pool.accTokenPerShare;
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (user.amount > 0) {
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getTokenBlockReward(pool.lastRewardBlock);
                uint256 tokenReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accTokenPerShare = accTokenPerShare.add(tokenReward.mul(1e12).div(lpSupply));
                return user.amount.mul(accTokenPerShare).div(1e12).sub(user.rewardDebt);
            }
            if (block.number == pool.lastRewardBlock) {
                return user.amount.mul(accTokenPerShare).div(1e12).sub(user.rewardDebt);
```

```
            }
        }
        return 0;
    }

    // Deposit LP tokens to Pool for MYT allocation.
    function deposit(uint256 _pid, uint256 _amount) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            depositTokenAndToken(_pid, _amount, msg.sender);
        } else {
            depositToken(_pid, _amount, msg.sender);
        }
    }

    function depositTokenAndToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pendingAmount = user.amount.mul(pool.accTokenPerShare).div(1e12).sub(user.rewardD
            if (pendingAmount > 0) {
                TransferHelper.safeTransfer(
                    address(myt),
                    _user,
                    pendingAmount
                );
            }
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            IMasterChefHeco(multLpChef).deposit(poolCorrespond[_pid], 0);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12)
            uint256 tokenPending = user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpR
            if (tokenPending > 0) {
                TransferHelper.safeTransfer(
                    multLpToken,
                    _user,
                    tokenPending
                );
            }
        }
        if (_amount > 0) {
            TransferHelper.safeTransferFrom(
                    address(pool.lpToken),
                    _user,
                     address(this),
                    _amount
                );
            if (pool.totalAmount == 0) {
                IMasterChefHeco(multLpChef).deposit(poolCorrespond[_pid], _amount);
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            } else {
                uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
                IMasterChefHeco(multLpChef).deposit(poolCorrespond[_pid], _amount);
                uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
                pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            }
        }
        user.rewardDebt = user.amount.mul(pool.accTokenPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }
```

```
    function depositToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pendingAmount = user.amount.mul(pool.accTokenPerShare).div(1e12).sub(user.rewardD
            if (pendingAmount > 0) {
                TransferHelper.safeTransfer(
                    address(myt),
                    _user,
                    pendingAmount
                );
            }
        }
        if (_amount > 0) {
             TransferHelper.safeTransferFrom(
                    address(pool.lpToken),
                    _user,
                    address(this), _amount
                );
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accTokenPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }

    // Withdraw LP tokens from MyTradePool.
    function withdraw(uint256 _pid, uint256 _amount) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            withdrawTokenAndToken(_pid, _amount, msg.sender);
        } else {
            withdrawToken(_pid, _amount, msg.sender);
        }
    }

    function withdrawTokenAndToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawTokenAndToken: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accTokenPerShare).div(1e12).sub(user.rewardDebt)
        if (pendingAmount > 0) {
            TransferHelper.safeTransfer(
                   address(myt),
                    _user,
                    pendingAmount
                );
        }
        if (_amount > 0) {
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            IMasterChefHeco(multLpChef).withdraw(poolCorrespond[_pid], _amount);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12)
            uint256 tokenPending = user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpR
            if (tokenPending > 0) {
                TransferHelper.safeTransfer(
                   address(multLpToken),
                    _user,
                    tokenPending
                );
            }
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
             TransferHelper.safeTransfer(
```

```
                address(pool.lpToken),
                _user,
                _amount
            );
        }
        user.rewardDebt = user.amount.mul(pool.accTokenPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }

    function withdrawToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawToken: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accTokenPerShare).div(1e12).sub(user.rewardDebt)
        if (pendingAmount > 0) {
            TransferHelper.safeTransfer(
                    address(myt),
                    _user,
                    pendingAmount
                );
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            TransferHelper.safeTransfer(
                    address(pool.lpToken),
                    _user,
                    _amount
                );
        }
        user.rewardDebt = user.amount.mul(pool.accTokenPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            emergencyWithdrawTokenAndToken(_pid, msg.sender);
        } else {
            emergencyWithdrawToken(_pid, msg.sender);
        }
    }

    function emergencyWithdrawTokenAndToken(uint256 _pid, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 amount = user.amount;
        uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
        IMasterChefHeco(multLpChef).withdraw(poolCorrespond[_pid], amount);
        uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
        pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div
        user.amount = 0;
        user.rewardDebt = 0;
        TransferHelper.safeTransfer(
                    address(pool.lpToken),
                    _user,
                    amount
                );
        pool.totalAmount = pool.totalAmount.sub(amount);
        emit EmergencyWithdraw(_user, _pid, amount);
    }

    function emergencyWithdrawToken(uint256 _pid, address _user) private {
```

```
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        TransferHelper.safeTransfer(
                    address(pool.lpToken),
                    _user,
                    amount
                );
        pool.totalAmount = pool.totalAmount.sub(amount);
        emit EmergencyWithdraw(_user, _pid, amount);
    }

    modifier notPause() {
        require(paused == false, "Mining has been suspended");
        _;
    }
}
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.8.0;
contract Ownable {
    address private _owner;

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

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor () {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     * @notice Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
```

```
        _owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0));
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
contract ReentrancyGuard {
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;
    uint256 private _status;
    constructor () {
        _status = _NOT_ENTERED;
    }
    modifier nonReentrant() {
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
        _status = _ENTERED;
        _;
        _status = _NOT_ENTERED;
    }
}
library SafeMath {
    /**
     * @dev Multiplies two unsigned integers, reverts on 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-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
     * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
```

```
     * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Adds two unsigned integers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}
interface IUniswapV2Factory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}
interface IMyTradeOrderBookExt{
    function liquidityPrice(
        address _fromTokenAddr,
        address _toTokenAddr,
        address _pairAddr,
        uint _reserve0,
        uint _reserve1
    )external;


    function cancelOrderWithNum(//按数量取消订单
        address _fromTokenAddr,// 卖出token地址
        address _toTokenAddr,// 买入token地址
        uint256 _orderIndex,// 具体订单号（目前是订单的唯一性标识）
        uint256 _num
    )external;

    function addOrder(
        address _maker,
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber,
        uint256 _orderIndex
    )external;
    function updateOrderInfo(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _orderIndex,
        uint256 _outNum,
        uint256 _inNum
    )external;
    function getOrderIndexsForMaker(
        address _fromTokenAddr,
        address _toTokenAddr,
```

```
        address _maker
    )external view returns(uint256[] memory cindexs);
    function getPageOrdersForMaker(// 分页获取所有订单号
        address _fromTokenAddr,// 卖出token地址
        address _toTokenAddr,// 买入token地址
        address _maker,
        uint256 _start,//开始位置
        uint256 _num//数量
    )external view returns(uint256[] memory indexs);
    function getOrderInfo(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _orderIndex
   )external view returns(uint256 _orderTime,uint256 _toTokenSum);
}
contract MyTradeOrderBookExt is Ownable,ReentrancyGuard,IMyTradeOrderBookExt{
    using SafeMath for uint;
    IUniswapV2Factory immutable public uniswapV2Factory;
    constructor(address _uniswapV2Factory) payable  {
        tokenPairExtArray.push();
        uniswapV2Factory=IUniswapV2Factory(_uniswapV2Factory);
    }
    receive() external payable {
    }
    fallback(bytes calldata _input) external payable returns (bytes memory _output){
    }

    struct TokenPairExt{
        mapping(uint256=> uint256) toTokenSumMap;// orderIndex=》toTokenSum
        mapping(uint256=> uint256) timeMap;// orderIndex=》time
        mapping(address => uint256[]) ordersForAddress;// 下单地址=》该地址对应的orderIndex
    }

    TokenPairExt[] tokenPairExtArray;// tokenPair数组
    mapping (address  => uint256) public tokenPairExtIndexMap;// token0地址=>tokenPairExt数组下标

    modifier onlyApproved(
        address payable addr
    ) {
        require(isApproved(address(this),addr));//必须经过该地址允许才能操作
        _;
    }
    // Mapping from owner to operator approvals
    mapping (address  => mapping (address  => bool)) private _operatorApprovals;
    /**
     *指定允许代理其对合约操作的权限的操作员
     */
    function setApproval(
        address to,
        bool approved
    ) public onlyOwner returns(bool) {
        _operatorApprovals[address(this)][to] = approved;
        return true;
    }
    function isApproved(
        address addr,
        address operator
    ) public view returns (bool) {
        if (addr == operator) {//如果是自己，默认允许
            return true;
        }
        return _operatorApprovals[addr][operator];
    }
    event LiquidityPrice(
        address indexed _fromTokenAddr,// 卖出token地址
        address indexed _toTokenAddr,// 买入token地址
```

```
        address indexed _pairAddr,
        uint _reserve0,
        uint _reserve1
    );
    function liquidityPrice(
        address _fromTokenAddr,
        address _toTokenAddr,
        address _pairAddr,
        uint _reserve0,
        uint _reserve1
    )onlyApproved(msg.sender) override public {
        LiquidityPrice(_fromTokenAddr,_toTokenAddr,_pairAddr,_reserve0,_reserve1);
    }
    event CancelOrderWithNum(
        address indexed _fromTokenAddr,// 卖出token地址
        address indexed _toTokenAddr,// 买入token地址
        uint256 indexed _orderIndex,// 具体订单号（目前是订单的唯一性标识）
        uint256 _num
    );
   function cancelOrderWithNum(//按数量取消订单
        address _fromTokenAddr,// 卖出token地址
        address _toTokenAddr,// 买入token地址
        uint256 _orderIndex,// 具体订单号（目前是订单的唯一性标识）
        uint256 _num
    )onlyApproved(msg.sender) override public {
        emit CancelOrderWithNum(_fromTokenAddr,_toTokenAddr,_orderIndex,_num);
    }
    event AddOrder(
        address indexed _maker,
        address indexed _fromTokenAddr,
        address _toTokenAddr,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber,
        uint256 indexed _orderIndex
    );

    function addOrder(
        address _maker,
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber,
        uint256 _orderIndex
    )onlyApproved(msg.sender) override public {
        address pairAddr=uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
        uint256 tokenPairIndex=tokenPairExtIndexMap[pairAddr];
        if(tokenPairIndex== 0){//如果交易对不存在就新增一个
            tokenPairExtArray.push();
            tokenPairIndex=tokenPairExtArray.length-1 ;
            tokenPairExtIndexMap[pairAddr]=tokenPairIndex;
        }
        tokenPairExtArray[tokenPairIndex].ordersForAddress[_maker].push(_orderIndex);
        tokenPairExtArray[tokenPairIndex].timeMap[_orderIndex]=block.timestamp;
        emit AddOrder(_maker,_fromTokenAddr,_toTokenAddr,_fromTokenNumber,_toTokenNumber,_orderIndex)
    }
    event UpdateOrderInfo(
        address indexed _fromTokenAddr,
        address indexed _toTokenAddr,
        uint256 indexed _orderIndex,
        uint256 _outNum,
        uint256 _inNum
    );
    function updateOrderInfo(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _orderIndex,
```

```
        uint256 _outNum,
        uint256 _inNum
    )onlyApproved(msg.sender) override public {
        address pairAddr=uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
        uint256 tokenPairIndex=tokenPairExtIndexMap[pairAddr];
        tokenPairExtArray[tokenPairIndex].toTokenSumMap[_orderIndex]=
        _outNum.add(tokenPairExtArray[tokenPairIndex].toTokenSumMap[_orderIndex]);
        tokenPairExtArray[tokenPairIndex].timeMap[_orderIndex]=block.timestamp;
        emit UpdateOrderInfo(_fromTokenAddr,_toTokenAddr,_orderIndex,_outNum,_inNum);
    }
    function getOrderIndexsForMaker(
        address _fromTokenAddr,
        address _toTokenAddr,
        address _maker
    )override public view returns(uint256[] memory cindexs){
        address pairAddr=uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
        uint256 tokenPairIndex=tokenPairExtIndexMap[pairAddr];
        return tokenPairExtArray[tokenPairIndex].ordersForAddress[_maker];
    }
    function getPageOrdersForMaker(// 分页获取所有订单号
        address _fromTokenAddr,// 卖出token地址
        address _toTokenAddr,// 买入token地址
        address _maker,
        uint256 _start,//开始位置
        uint256 _num//数量
    )override public view returns(uint256[] memory indexs){
        address pairAddr=uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
        uint256 tokenPairIndex=tokenPairExtIndexMap[pairAddr];
        indexs=new uint256[](_num);
        uint256 cl=tokenPairExtArray[tokenPairIndex].ordersForAddress[_maker].length;
        for(uint256 i=_start;i<cl&&i<_start+_num;i++){
            indexs[i-_start]=tokenPairExtArray[tokenPairIndex].ordersForAddress[_maker][i];
        }
    }
    function getOrderInfo(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _orderIndex
   )override public view returns(uint256 _orderTime,uint256 _toTokenSum){
       address pairAddr=uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
       uint256 tokenPairIndex=tokenPairExtIndexMap[pairAddr];
       return (tokenPairExtArray[tokenPairIndex].timeMap[_orderIndex],tokenPairExtArray[tokenPairInde
   }
}// SPDX-License-Identifier: MIT
pragma solidity =0.6.6;

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

    function feeTo() external view returns (address);

    function feeToSetter() external view returns (address);

    function feeToRate() external view returns (uint256);

    function initCodeHash() external view returns (bytes32);

    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;

    function setFeeToRate(uint256) external;

    function setInitCodeHash(bytes32) external;

    function sortTokens(address tokenA, address tokenB) external pure returns (address token0, addres

    function pairFor(address tokenA, address tokenB) external view returns (address pair);

    function getReserves(address tokenA, address tokenB) external view returns (uint256 reserveA, uin

    function quote(uint256 amountA, uint256 reserveA, uint256 reserveB) external pure returns (uint25

    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) external view retu

    function getAmountIn(uint256 amountOut, uint256 reserveIn, uint256 reserveOut) external view retu

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

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

interface IMytradePair {
    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, by

    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 blockTim

    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 price(address token, uint256 baseDecimal) external view returns (uint256);

    function initialize(address, address) external;
}

interface IMytradeRouter {
    function factory() external pure returns (address);

    function WHT() external pure returns (address);

    function swapMining() 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 deadl
    external
    payable
    returns (uint[] memory amounts);

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

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

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

```
    function quote(uint256 amountA, uint256 reserveA, uint256 reserveB) external view returns (uint25

    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) external view retu

    function getAmountIn(uint256 amountOut, uint256 reserveIn, uint256 reserveOut) external view retu

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

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

    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;
}

interface ISwapMining {
    function swap(address account, address input, address output, uint256 amount) external returns (b
}

contract Ownable {
    address private _owner;

    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

    function owner() public view returns (address) {
```

```
        return _owner;
    }

    function isOwner(address account) public view returns (bool) {
        return account == _owner;
    }

    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }

    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }


    modifier onlyOwner() {
        require(isOwner(msg.sender), "Ownable: caller is not the owner");
        _;
    }

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

interface IERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external view returns (string memory);

    function symbol() external view returns (string memory);

    function decimals() external view 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);
}

interface IWHT {
    function deposit() external payable;

    function transfer(address to, uint value) external returns (bool);

    function withdraw(uint) external;
}

contract MytradeRouter is IMytradeRouter, Ownable {
    using SafeMath for uint256;

    address public immutable override factory;
```

```
    address public immutable override WHT;
    address public override swapMining;
    // orderbook address
    address public orderbook;
    modifier onlyOrderbook() {
        require(msg.sender == orderbook, "MytradeRouter: caller is not the orderbook");
        _;
    }
    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'MytradeRouter: EXPIRED');
        _;
    }
    function setOrderbook(address _orderbook) external {
        if(orderbook==address(0)){
            orderbook = _orderbook;
        }
    }
    constructor(address _factory, address _WHT) public {
        factory = _factory;
        WHT = _WHT;
    }

    receive() external payable {
        assert(msg.sender == WHT);
        // only accept HT via fallback from the WHT contract
    }

    function pairFor(address tokenA, address tokenB) public view returns (address pair){
        pair = IMytradeFactory(factory).pairFor(tokenA, tokenB);
    }

    function setSwapMining(address _swapMininng) public onlyOwner {
        swapMining = _swapMininng;
    }

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal virtual returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        if (IMytradeFactory(factory).getPair(tokenA, tokenB) == address(0)) {
            IMytradeFactory(factory).createPair(tokenA, tokenB);
        }
        (uint reserveA, uint reserveB) = IMytradeFactory(factory).getReserves(tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = IMytradeFactory(factory).quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'MytradeRouter: INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = IMytradeFactory(factory).quote(amountBDesired, reserveB, reserv
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'MytradeRouter: INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function addLiquidity(
```

```
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity)
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin
        address pair = pairFor(tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IMytradePair(pair).mint(to);
    }

    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint amountToken, uint amountETH, u
        (amountToken, amountETH) = _addLiquidity(
            token,
            WHT,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = pairFor(token, WHT);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWHT(WHT).deposit{value : amountETH}();
        assert(IWHT(WHT).transfer(pair, amountETH));
        liquidity = IMytradePair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender, msg.value - amountETH);
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
        address pair = pairFor(tokenA, tokenB);
        IMytradePair(pair).transferFrom(msg.sender, pair, liquidity);
        // send liquidity to pair
        (uint amount0, uint amount1) = IMytradePair(pair).burn(to);
        (address token0,) = IMytradeFactory(factory).sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'MytradeRouter: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'MytradeRouter: INSUFFICIENT_B_AMOUNT');
    }

    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
```

```
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountToken, uint amountETH) {
        (amountToken, amountETH) = removeLiquidity(
            token,
            WHT,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWHT(WHT).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, 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 virtual override returns (uint amountA, uint amountB) {
        address pair = pairFor(tokenA, tokenB);
        uint value = approveMax ? uint(- 1) : liquidity;
        IMytradePair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin, amountBMin, to, d
    }

    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountToken, uint amountETH) {
        address pair = pairFor(token, WHT);
        uint value = approveMax ? uint(- 1) : liquidity;
        IMytradePair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountETH) = removeLiquidityETH(token, liquidity, amountTokenMin, amountETHMin,
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountETH) {
        (, amountETH) = removeLiquidity(
            token,
            WHT,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
```

```
        TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));
        IWHT(WHT).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountETH) {
        address pair = pairFor(token, WHT);
        uint value = approveMax ? uint(- 1) : liquidity;
        IMytradePair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountETHMin, to, deadline
        );
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal onlyOrderbook
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = IMytradeFactory(factory).sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(msg.sender, input, output, amountOut);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amountOut,
            address to = i < path.length - 2 ? pairFor(output, path[i + 2]) : _to;
            IMytradePair(pairFor(input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }

    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = IMytradeFactory(factory).getAmountsOut(amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'MytradeRouter: INSUFFICIENT_OUTPUT_AMOU
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }

    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = IMytradeFactory(factory).getAmountsIn(amountOut, path);
        require(amounts[0] <= amountInMax, 'MytradeRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
```

```
        );
        _swap(amounts, path, to);
    }

    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadl
    external
    virtual
    override
    payable
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(path[0] == WHT, 'MytradeRouter: INVALID_PATH');
        amounts = IMytradeFactory(factory).getAmountsOut(msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'MytradeRouter: INSUFFICIENT_OUTPUT_AMOU
        IWHT(WHT).deposit{value : amounts[0]}();
        assert(IWHT(WHT).transfer(pairFor(path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
    }

    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address
    external
    virtual
    override
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WHT, 'MytradeRouter: INVALID_PATH');
        amounts = IMytradeFactory(factory).getAmountsIn(amountOut, path);
        require(amounts[0] <= amountInMax, 'MytradeRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWHT(WHT).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address
    external
    virtual
    override
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WHT, 'MytradeRouter: INVALID_PATH');
        amounts = IMytradeFactory(factory).getAmountsOut(amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'MytradeRouter: INSUFFICIENT_OUTPUT_AMOU
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWHT(WHT).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline
    external
    virtual
    override
    payable
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(path[0] == WHT, 'MytradeRouter: INVALID_PATH');
        amounts = IMytradeFactory(factory).getAmountsIn(amountOut, path);
```

```
        require(amounts[0] <= msg.value, 'MytradeRouter: EXCESSIVE_INPUT_AMOUNT');
        IWHT(WHT).deposit{value : amounts[0]}();
        assert(IWHT(WHT).transfer(pairFor(path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender, msg.value - amounts[0]
    }

    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
    function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal virtual
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = IMytradeFactory(factory).sortTokens(input, output);
            IMytradePair pair = IMytradePair(pairFor(input, output));
            uint amountInput;
            uint amountOutput;
            {// scope to avoid stack too deep errors
                (uint reserve0, uint reserve1,) = pair.getReserves();
                (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) : (r
                amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput);
                amountOutput = IMytradeFactory(factory).getAmountOut(amountInput, reserveInput, reser
            }
            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(msg.sender, input, output, amountOutput);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) : (amountO
            address to = i < path.length - 2 ? pairFor(output, path[i + 2]) : _to;
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) {
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amountIn
        );
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'MytradeRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
    external
    virtual
    override
    payable
    ensure(deadline)
    {
        require(path[0] == WHT, 'MytradeRouter: INVALID_PATH');
        uint amountIn = msg.value;
        IWHT(WHT).deposit{value : amountIn}();
        assert(IWHT(WHT).transfer(pairFor(path[0], path[1]), amountIn));
```

```
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'MytradeRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
    external
    virtual
    override
    ensure(deadline)
    {
        require(path[path.length - 1] == WHT, 'MytradeRouter: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20(WHT).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'MytradeRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        IWHT(WHT).withdraw(amountOut);
        TransferHelper.safeTransferETH(to, amountOut);
    }

    // **** LIBRARY FUNCTIONS ****
    function quote(uint256 amountA, uint256 reserveA, uint256 reserveB) public view override returns
        return IMytradeFactory(factory).quote(amountA, reserveA, reserveB);
    }

    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) public view overri
        return IMytradeFactory(factory).getAmountOut(amountIn, reserveIn, reserveOut);
    }

    function getAmountIn(uint256 amountOut, uint256 reserveIn, uint256 reserveOut) public view overri
        return IMytradeFactory(factory).getAmountIn(amountOut, reserveIn, reserveOut);
    }

    function getAmountsOut(uint256 amountIn, address[] memory path) public view override returns (uin
        return IMytradeFactory(factory).getAmountsOut(amountIn, path);
    }

    function getAmountsIn(uint256 amountOut, address[] memory path) public view override returns (uin
        return IMytradeFactory(factory).getAmountsIn(amountOut, path);
    }

}
/**
 * @title SafeMath
 * @dev Unsigned math operations with safety checks that revert on error.
 */
library SafeMath {
    /**
     * @dev Multiplies two unsigned integers, reverts on 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-solidity/pull/522
        if (a == 0) {
```

```
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
     * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Adds two unsigned integers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}


// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: APPROVE_F
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }
```

```
    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }

    function safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value : value}(new bytes(0));
        require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
    }
}// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.8.0;
//import "hardhat/console.sol";
contract Ownable {
    address private _owner;

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

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor () {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     * @notice Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function transferOwnership(address newOwner) public onlyOwner {
```

```
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0));
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
contract ReentrancyGuard {
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;
    uint256 private _status;
    constructor () {
        _status = _NOT_ENTERED;
    }
    modifier nonReentrant() {
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
        _status = _ENTERED;
        _;
        _status = _NOT_ENTERED;
    }
}
library SafeMath {
    /**
     * @dev Multiplies two unsigned integers, reverts on 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-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
     * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }
```

```
    /**
     * @dev Adds two unsigned integers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}
interface IUniswapV2Factory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}
// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'TransferHelper: MyTradeOrderBook ETH_TRANSFER_FAILED');
    }
}
interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns (uint256);

}
contract MyTradeOrderMining is Ownable,ReentrancyGuard{
    using SafeMath for uint;
    IUniswapV2Factory immutable public uniswapV2Factory;
    // Mapping from owner to operator approvals
```

```
    mapping (address  => mapping (address  => bool)) private _operatorApprovals;
    address public userRewardToken;
    address public totalPoolAddr;
    uint public orderMiningPerBlock;
    // The block number when mining starts.
    uint256 public startBlock;

    UserInfo[] public userInfoArray;// UserInfo数组
    mapping (address  => uint256) public userInfoIndexMap;// maker地址=>userInfoArray数组下标

    struct UserInfo {
        address maker;
        uint256 withdrawEndBlock;
        uint256 addRewardEndBlock;
        uint256 reward;
        uint256 rewardSum;
    }

    event UpdateUserReward(
        address indexed orderMaker,
        uint256 addRewardEndBlock,
        uint256 addReward
    );
    constructor(
        address _uniswapV2Factory,
        uint _startBlock
    ) payable  {
        startBlock = _startBlock;
        uniswapV2Factory=IUniswapV2Factory(_uniswapV2Factory);
        userInfoArray.push();
    }
    receive() external payable {
    }
    fallback(bytes calldata _input) external payable returns (bytes memory _output){
    }
     modifier onlyApproved(
        address payable addr
    ) {
        require(isApproved(address(this),addr));//必须经过该地址允许才能操作
        _;
    }
    /**
     *指定允许代理其对合约操作的权限的操作员
     */
    function setApproval(
        address to,
        bool approved
    ) public onlyOwner returns(bool) {
        _operatorApprovals[address(this)][to] = approved;
        return true;
    }
    function isApproved(
        address addr,
        address operator
    ) public view returns (bool) {
        if (addr == operator) {//如果是自己，默认允许
            return true;
        }
        return _operatorApprovals[addr][operator];
    }

    function setUserRewardToken(
        address _userRewardToken
    ) onlyApproved(msg.sender) public returns(bool) {
        userRewardToken=_userRewardToken;
        return true;
```

```
    }
    function setTotalPoolAddr(
        address _totalPoolAddr
    ) onlyApproved(msg.sender) public returns(bool) {
        totalPoolAddr=_totalPoolAddr;
        return true;
    }
    function setOrderMiningPerBlock(
        uint _orderMiningPerBlock
    ) onlyApproved(msg.sender) public returns(bool) {
        orderMiningPerBlock=_orderMiningPerBlock;
        return true;
    }
    function updateOrderMiningNumByOuter() public nonReentrant() returns (bool){
        return updateOrderMiningNum();
    }
    function setOrderMiningStartBlock(
        uint _startBlock
    ) onlyApproved(msg.sender) public returns(bool) {
        startBlock=_startBlock;
        return true;
    }
    function updateUserReward(
        address _orderMaker,
        uint256 _addRewardOldEndBlock,
        uint256 _addRewardCurEndBlock,
        uint256 _addReward
    ) public onlyApproved(msg.sender) returns (bool){
        uint256 userInfoIndex=userInfoIndexMap[_orderMaker];
        if(userInfoIndex== 0){//如果交易对不存在就新增一个
            userInfoArray.push();
            userInfoIndex=userInfoArray.length-1 ;
            userInfoIndexMap[_orderMaker]=userInfoIndex;
            userInfoArray[userInfoIndex].maker=_orderMaker;
        }
        //防止重复
        require(_addRewardOldEndBlock==userInfoArray[userInfoIndex].addRewardEndBlock,"data error");
        userInfoArray[userInfoIndex].reward=userInfoArray[userInfoIndex].reward.add(_addReward);
        userInfoArray[userInfoIndex].rewardSum=userInfoArray[userInfoIndex].rewardSum.add(_addReward)
        userInfoArray[userInfoIndex].addRewardEndBlock=_addRewardCurEndBlock;
        emit UpdateUserReward(_orderMaker,_addRewardCurEndBlock,_addReward);
        return true;
    }

    function updateOrderMiningNum() public returns (bool){
        if(block.number > startBlock){
            uint256 reward=block.number.sub(startBlock).mul(orderMiningPerBlock);
            startBlock=block.number;
            TransferHelper.safeTransferFrom(
                userRewardToken,
                totalPoolAddr,
                address(this),
                reward
            );
        }
        return true;
    }


    function withdrawUserReward() public nonReentrant returns (bool){
        uint256 userInfoIndex=userInfoIndexMap[msg.sender];
        require(userInfoIndex!= 0);
        require(updateOrderMiningNum());
        require(userInfoArray[userInfoIndex].reward>0);
        TransferHelper.safeTransfer(
            userRewardToken,
```

```
            msg.sender,
            userInfoArray[userInfoIndex].reward
        );
        userInfoArray[userInfoIndex].reward=0;
        userInfoArray[userInfoIndex].withdrawEndBlock=userInfoArray[userInfoIndex].addRewardEndBlock;
        return true;
    }
}// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.8.0;

/**
 * @title SafeMath
 * @dev Unsigned math operations with safety checks that revert on error.
 */
library SafeMath {
    /**
     * @dev Multiplies two unsigned integers, reverts on 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-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b,"mul");

        return c;
    }

    /**
     * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0,"div");
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a,"sub");
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Adds two unsigned integers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a,"add");

        return c;
    }

}
/**
```

```
 * @title ERC20 interface
 * @dev see https://eips.ethereum.org/EIPS/eip-20
 */
interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);
    function transferFrom(address from, address to, uint256 value) external returns (bool);
    function balanceOf(address who) external view returns (uint256);

}
interface IUniswapV2Factory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}
interface IUniswapV2Pair {
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
}
/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
    address private _owner;

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

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor () {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(),"onlyOwner");
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     * @notice Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
```

```
        _owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0),"incorrect address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
contract ReentrancyGuard {
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;
    uint256 private _status;
    constructor () {
        _status = _NOT_ENTERED;
    }
    modifier nonReentrant() {
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
        _status = _ENTERED;
        _;
        _status = _NOT_ENTERED;
    }
}
interface IWETH {
    function deposit() external payable;

    function transfer(address to, uint value) external returns (bool);

    function withdraw(uint) external;
}
// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'TransferHelper: MyTradeOrderBook ETH_TRANSFER_FAILED');
    }
```

```
}

interface IMyTradeOrderBookExt{
    function liquidityPrice(
        address _fromTokenAddr,
        address _toTokenAddr,
        address _pairAddr,
        uint _reserve0,
        uint _reserve1
    )external;
    function cancelOrderWithNum(//按数量取消订单
        address _fromTokenAddr,// 卖出token地址
        address _toTokenAddr,// 买入token地址
        uint256 _orderIndex,// 具体订单号（目前是订单的唯一性标识）
        uint256 _num
    )external;

    function addOrder(
        address _maker,
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber,
        uint256 _orderIndex
    )external;
    function updateOrderInfo(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _orderIndex,
        uint256 _outNum,
        uint256 _inNum
    )external;
    function getOrderIndexsForMaker(
        address _fromTokenAddr,
        address _toTokenAddr,
        address _maker
    )external view returns(uint256[] memory cindexs);
    function getOrderInfo(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _orderIndex
   )external view returns(uint256 _orderTime,uint256 _toTokenSum);
}
interface IMyTradeOrderMining{
    function updateOrderMiningNumByOuter() external returns (bool);
}
interface ISwapMining {
    function swap(address account, address input, address output, uint256 amount) external returns (b
}
library OrderBookHelper{
    using SafeMath for uint;

   // given an input amount of an asset and pair reserves, returns the maximum output amount of the o
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint
        uint amountInWithFee = amountIn.mul(997);
        uint numerator = amountInWithFee.mul(reserveOut);
        uint denominator = reserveIn.mul(1000).add(amountInWithFee);
        amountOut = numerator / denominator;
    }
     // babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian
    function sqrt(uint256 y) internal pure returns (uint256 z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
```

```
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
    function getInAmount(
        uint256 fromNum,
        uint256 toNum,
        uint256 reserveA,
        uint256 reserveB
    ) internal pure returns(uint256 z){
        uint256 p=reserveA.mul(reserveB).mul(fromNum).mul(1000)/997/toNum;
        uint256 q=reserveA.mul(reserveA).mul(8973)/3964107892;
        uint256 x=sqrt(p.add(q));

        uint256 y=reserveA.mul(1997)/1994;
        if(x>y){
            z=x.sub(y).add(1);
        }else{
            z=0;
        }
    }
    function joinNumber(
        uint256 _number,
        uint256[] memory narray
    )internal pure returns(uint256[] memory){
        if(_number==0){
            return narray;
        }
        uint256 nl=narray.length;
        uint256[] memory narray1=new uint256[](nl+1);
        for(uint256 i=0;i<nl;i++){
            narray1[i]=narray[i];
        }
        narray1[nl]=_number;
        return narray1;
    }
    function sortTokens(address tokenA, address tokenB) internal pure returns (address token0, addres
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
    }
}
contract MyTradeOrderBook is Ownable,ReentrancyGuard{
    using SafeMath for uint;
    IUniswapV2Factory immutable public uniswapV2Factory;
    address immutable public WETH;
    address public feeAddr;
    uint256 constant UINT256_MAX = ~uint256(0);
    IMyTradeOrderBookExt myTradeOrderBookExt;
    address public myTradeOrderMining;
    address public swapMining;
    function setMyTradeOrderMining(address _myTradeOrderMining) public onlyOwner {
        myTradeOrderMining = _myTradeOrderMining;
    }
    address public forPreDiposit;
     function setForPreDiposit(address _forPreDiposit) public onlyOwner {
        forPreDiposit = _forPreDiposit;
    }
    function setSwapMining(address _swapMininng) public onlyOwner {
        swapMining = _swapMininng;
    }
    //最小数量限额：0.1,可外部设置
    mapping (address  => uint) minLimitMap;
    /**
     *设置最小允许的数
     */
```

```
    function setMinLimit(
        address _tokenAddr,
        uint _minLimit
    ) onlyOwner public returns(bool) {
        minLimitMap[_tokenAddr] = _minLimit;
        return true;
    }

    struct Order{
        address maker;
        address fromTokenAddr;
        address toTokenAddr;
        uint256 remainNumber;
        uint256 fromTokenNumber;// 代币挂单金额
        uint256 toTokenNumber;// 意向代币目标金额
    }
    struct TokenPair{
        uint256 orderMaxIndex;
        mapping(address=> uint256) lastIndex;
        mapping(uint256=> Order) orderMap;// orderIndex=》Order
        mapping(uint256=> uint256) orderNextSequence;// 价格低的orderIndex=》价格高的orderIndex
        mapping(uint256=> uint256) orderPreSequence;// 价格高的orderIndex=》价格低的orderIndex
    }
    mapping (address=> mapping (uint  => uint8)) isForEth;
    TokenPair[] tokenPairArray;// tokenPair数组
    mapping (address  => uint256) tokenPairIndexMap;// tokenPairAddr=>tokenPair数组下标
    address immutable public flashLoan;
    constructor(
        address _WETH,
        address _uniswapV2Factory
    ) payable  {
        WETH=_WETH;
        feeAddr=msg.sender;
        tokenPairArray.push();
        uniswapV2Factory=IUniswapV2Factory(_uniswapV2Factory);
        flashLoan=address(new FlashLoan(msg.sender));
    }
    function safeApproveFlashLoan(
        address tokenA
    )public{
        TransferHelper.safeApprove(
            tokenA,
            flashLoan,
            UINT256_MAX
        );
    }
    function setMyTradeOrderBookExtAddr(
        address _myTradeOrderBookExtAddr
    ) onlyOwner public returns(bool) {
        myTradeOrderBookExt=IMyTradeOrderBookExt(_myTradeOrderBookExtAddr);
        return true;
    }
    receive() external payable {
        if(msg.value>0&&msg.sender!=WETH){
            IWETH(WETH).deposit{value : msg.value}();
        }
    }
    fallback(bytes calldata _input) external payable returns (bytes memory _output){
    }
    function setFeeAddr(address _feeAddr)public onlyOwner {
        feeAddr=_feeAddr;
    }
    mapping (address  => uint256) public allUserDiposit;//所有用户存款代币数量
    mapping (address  => mapping (address  => uint256)) public userDiposit;
    function deposit(address _token,uint _num) public {
        TransferHelper.safeTransferFrom(
```

```
            _token,
            msg.sender,
            address(this),
            _num
        );
        userDiposit[msg.sender][_token] = userDiposit[msg.sender][_token].add(_num);
        allUserDiposit[_token]=allUserDiposit[_token].add(_num);
    }
    function withdraw(address _token,uint _num) public {
        require(userDiposit[msg.sender][_token]>=_num,"Insufficient Number");
        TransferHelper.safeTransfer(
                _token,
                msg.sender,
                _num
            );
        userDiposit[msg.sender][_token] = userDiposit[msg.sender][_token].sub(_num);
        allUserDiposit[_token]=allUserDiposit[_token].sub(_num);
    }

    function addOrderWithPreDiposit(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _targetOrderIndex,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber
    )public nonReentrant returns(uint256 reserveNum,uint256 orderIndex) {
        require(msg.sender==forPreDiposit);
        require(userDiposit[msg.sender][_fromTokenAddr]>=_fromTokenNumber,"Insufficient Balance");
        require(_fromTokenNumber>=minLimitMap[_fromTokenAddr],"min limit");
        userDiposit[msg.sender][_fromTokenAddr] = userDiposit[msg.sender][_fromTokenAddr].sub(_fromTo
        (reserveNum,orderIndex)=_addOrder(
            msg.sender,
            _fromTokenAddr,
            _toTokenAddr,
            _targetOrderIndex,
            _fromTokenNumber,
            _toTokenNumber
        );
    }
    function cancelOrderForNumWithPreDiposit(//按数量取消订单
        address _fromTokenAddr,// 卖出token地址
        address _toTokenAddr,// 买入token地址
        uint256 _orderIndex,// 具体订单号（目前是订单的唯一性标识）
        uint256 _num
    )public nonReentrant returns(bool) {
        _cancelOrderForNum(_fromTokenAddr,_toTokenAddr,_orderIndex,_num);
        userDiposit[msg.sender][_fromTokenAddr] = userDiposit[msg.sender][_fromTokenAddr].add(_num);
        return true;
    }
    function cancelOrderForNum(//按数量取消订单
        address _fromTokenAddr,// 卖出token地址
        address _toTokenAddr,// 买入token地址
        uint256 _orderIndex,// 具体订单号（目前是订单的唯一性标识）
        uint256 _num
    )public nonReentrant returns(bool) {
        _cancelOrderForNum(_fromTokenAddr,_toTokenAddr,_orderIndex,_num);
        allUserDiposit[_fromTokenAddr]=allUserDiposit[_fromTokenAddr].sub(_num);
        if(_fromTokenAddr==WETH){
           IWETH(WETH).withdraw(_num);
           TransferHelper.safeTransferETH(
              msg.sender,
              _num
           );
        }else{
           TransferHelper.safeTransfer(
                _fromTokenAddr,
```

```
                msg.sender,
                _num
            );
        }
        if(myTradeOrderMining!=address(0)){
            IMyTradeOrderMining(myTradeOrderMining).updateOrderMiningNumByOuter();
        }
        return true;
    }
    function _cancelOrderForNum(//按数量取消订单
        address _fromTokenAddr,// 卖出token地址
        address _toTokenAddr,// 买入token地址
        uint256 _orderIndex,// 具体订单号（目前是订单的唯一性标识）
        uint256 _num
    )internal{
        address pairAddr=uniswapV2Factory.getPair(_fromTokenAddr,_toTokenAddr);
        uint256 tokenPairIndex=tokenPairIndexMap[pairAddr];
        require(tokenPairIndex!=0,"tokenPair not exist");
        TokenPair storage _tokenPair=tokenPairArray[tokenPairIndex];
        require(_tokenPair.orderMap[_orderIndex].maker==msg.sender,"invalid maker");
        require(_orderIndex!=0,"order not exist");
        uint256 remainNumber=_tokenPair.orderMap[_orderIndex].remainNumber;
        require(remainNumber>=_num,"Insufficient RemainNumber");
        if(remainNumber==_num){
            if(_tokenPair.lastIndex[_fromTokenAddr]==_orderIndex){
                _tokenPair.lastIndex[_fromTokenAddr]=_tokenPair.orderNextSequence[_orderIndex];
                _tokenPair.orderPreSequence[_tokenPair.lastIndex[_fromTokenAddr]]=0;
                _tokenPair.orderNextSequence[_orderIndex]=0;
            }else{
                uint256 orderPreIndex=_tokenPair.orderPreSequence[_orderIndex];
                require(orderPreIndex!=0,"no preIndex");
                require(_tokenPair.orderMap[_orderIndex].fromTokenAddr==_fromTokenAddr,"invalid fromT
                uint256 orderNextIndex=_tokenPair.orderNextSequence[_orderIndex];
                _tokenPair.orderNextSequence[orderPreIndex]=orderNextIndex;
                if(orderNextIndex!=0){
                    _tokenPair.orderPreSequence[orderNextIndex]=orderPreIndex;
                }
                _tokenPair.orderPreSequence[_orderIndex]=0;
                _tokenPair.orderNextSequence[_orderIndex]=0;
            }
        }else{
            _tokenPair.orderMap[_orderIndex].remainNumber=remainNumber.sub(_num);
        }

        (uint256 reserveA,uint256 reserveB)=getReserves(_fromTokenAddr, _toTokenAddr);
        myTradeOrderBookExt.liquidityPrice(_fromTokenAddr,_toTokenAddr,pairAddr,reserveA,reserveB);
        myTradeOrderBookExt.cancelOrderWithNum(_fromTokenAddr,_toTokenAddr,_orderIndex,_num);
    }
    function _orderIndexSequence(
        uint256 _targetOrderIndex,
        uint256 _orderIndex,
        TokenPair storage _tokenPair,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber
    )internal{
        if(_fromTokenNumber.mul(_tokenPair.orderMap[_targetOrderIndex].toTokenNumber)<=
            _tokenPair.orderMap[_targetOrderIndex].fromTokenNumber.mul(_toTokenNumber)){
            uint256 orderNextSequence=_tokenPair.orderNextSequence[_targetOrderIndex];
            if(orderNextSequence==0){
                _tokenPair.orderNextSequence[_targetOrderIndex]=_orderIndex;
                _tokenPair.orderPreSequence[_orderIndex]=_targetOrderIndex;
            }else{
                if(_fromTokenNumber.mul(_tokenPair.orderMap[orderNextSequence].toTokenNumber)<=
                    _tokenPair.orderMap[orderNextSequence].fromTokenNumber.mul(_toTokenNumber)){
                    _orderIndexSequence(orderNextSequence,
                        _orderIndex,_tokenPair,_fromTokenNumber,_toTokenNumber);
```

```
                }else{
                    _tokenPair.orderPreSequence[_orderIndex]=_targetOrderIndex;
                    _tokenPair.orderNextSequence[_targetOrderIndex]=_orderIndex;
                    _tokenPair.orderPreSequence[orderNextSequence]=_orderIndex;
                    _tokenPair.orderNextSequence[_orderIndex]=orderNextSequence;
                }
            }
        }else{
            uint256 orderPreIndex=_tokenPair.orderPreSequence[_targetOrderIndex];
            if(orderPreIndex==0){
                _tokenPair.orderPreSequence[_targetOrderIndex]=_orderIndex;
                _tokenPair.orderNextSequence[_orderIndex]=_targetOrderIndex;
            }else{
                if(_fromTokenNumber.mul(_tokenPair.orderMap[orderPreIndex].toTokenNumber)>=
                    _tokenPair.orderMap[orderPreIndex].fromTokenNumber.mul(_toTokenNumber)){
                    _orderIndexSequence(orderPreIndex,
                        _orderIndex,_tokenPair,_fromTokenNumber,_toTokenNumber);
                }else{
                    _tokenPair.orderNextSequence[_orderIndex]=_targetOrderIndex;
                    _tokenPair.orderPreSequence[_targetOrderIndex]=_orderIndex;
                    _tokenPair.orderNextSequence[orderPreIndex]=_orderIndex;
                    _tokenPair.orderPreSequence[_orderIndex]=orderPreIndex;
                }
            }
        }
    }

    function addOrderWithETH(
        address _maker,
        address _toTokenAddr,
        uint256 _targetOrderIndex,
        uint256 _toTokenNumber
    )public payable nonReentrant returns(uint256 reserveNum,uint256 orderIndex) {
        require(msg.value>minLimitMap[WETH],"min limit");
        IWETH(WETH).deposit{value : msg.value}();
        (reserveNum,orderIndex)=_addOrder(_maker,WETH,_toTokenAddr,_targetOrderIndex,msg.value,_toTok
    }
    function addOrderForETH(
        address _maker,
        address _fromTokenAddr,
        uint256 _targetOrderIndex,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber
    )public returns(uint256 reserveNum,uint256 orderIndex) {
        address pairAddr=uniswapV2Factory.getPair(_fromTokenAddr,WETH);
        require(pairAddr!=address(0),"pairAddr not exist");
        uint256 tokenPairIndex=tokenPairIndexMap[pairAddr];
        if(tokenPairIndex== 0){//如果交易对不存在就新增一个
            tokenPairIndex=tokenPairArray.length;
            tokenPairArray.push();
            tokenPairIndexMap[pairAddr]=tokenPairIndex;
        }
        isForEth[_fromTokenAddr][tokenPairArray[tokenPairIndex].orderMaxIndex.add(1)]=2;
        (reserveNum,orderIndex)=addOrder(
            _maker,
            _fromTokenAddr,
            WETH,
            _targetOrderIndex,
            _fromTokenNumber,
            _toTokenNumber
        );
    }
    function addOrder(
        address _maker,
        address _fromTokenAddr,
        address _toTokenAddr,
```

```
        uint256 _targetOrderIndex,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber
    )public nonReentrant returns(uint256 reserveNum,uint256 orderIndex) {
        require(_fromTokenNumber>minLimitMap[_fromTokenAddr],"min limit");
        TransferHelper.safeTransferFrom(
            _fromTokenAddr,
            msg.sender,
            address(this),
            _fromTokenNumber
        );
        (reserveNum,orderIndex)=_addOrder(_maker,_fromTokenAddr,
            _toTokenAddr,_targetOrderIndex,_fromTokenNumber,_toTokenNumber
        );
    }
    function _addOrder(
        address _maker,
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _targetOrderIndex,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber
    )internal returns(
        uint256 reserveNum,
        uint256 orderIndex
    ) {
        address pairAddr=uniswapV2Factory.getPair(_fromTokenAddr,_toTokenAddr);
        require(pairAddr!=address(0),"pairAddr not exist");
        uint256 tokenPairIndex=tokenPairIndexMap[pairAddr];
        if(tokenPairIndex== 0){//如果交易对不存在就新增一个
            tokenPairIndex=tokenPairArray.length;
            tokenPairArray.push();
            tokenPairIndexMap[pairAddr]=tokenPairIndex;
        }
        TokenPair storage _tokenPair=tokenPairArray[tokenPairIndex];
        orderIndex=_tokenPair.orderMaxIndex.add(1);
        _tokenPair.orderMaxIndex=orderIndex;

        uint256 lastIndex=_tokenPair.lastIndex[_fromTokenAddr];
        if(lastIndex!=0){
            if(_targetOrderIndex==0){
                _targetOrderIndex=lastIndex;
            }else if(
                _tokenPair.orderNextSequence[_targetOrderIndex]==0&&
                _tokenPair.orderPreSequence[_targetOrderIndex]==0
            ){
                _targetOrderIndex=lastIndex;
            }else if(_tokenPair.orderMap[_targetOrderIndex].fromTokenAddr!=_fromTokenAddr){
                _targetOrderIndex=lastIndex;
            }
            _orderIndexSequence(_targetOrderIndex,orderIndex,_tokenPair,
                _fromTokenNumber,_toTokenNumber
            );
        }
        myTradeOrderBookExt.addOrder(
            _maker,
            _fromTokenAddr,
            _toTokenAddr,
            _fromTokenNumber,
            _toTokenNumber,
            orderIndex);
        Order memory order=Order(_maker,_fromTokenAddr,_toTokenAddr,
           _fromTokenNumber,_fromTokenNumber,_toTokenNumber);
        _tokenPair.orderMap[orderIndex]=order;
        (uint256 reserveA,uint256 reserveB)=getReserves(_fromTokenAddr, _toTokenAddr);
        if(_tokenPair.orderPreSequence[orderIndex]==0){
```

```
            _tokenPair.lastIndex[order.fromTokenAddr]=orderIndex;
            checkTrade(_tokenPair,orderIndex,order,reserveA,reserveB);
            reserveNum=_tokenPair.orderMap[orderIndex].remainNumber;
            if(reserveNum==0){
                uint _nextIndex=_tokenPair.orderNextSequence[orderIndex];
                _tokenPair.lastIndex[order.fromTokenAddr]=_nextIndex;
                if(_nextIndex!=0){
                    _tokenPair.orderNextSequence[orderIndex]=0;
                    _tokenPair.orderPreSequence[_nextIndex]=0;
                }
            }else{
                allUserDiposit[order.fromTokenAddr]=allUserDiposit[order.fromTokenAddr].add(reserveNu
            }
            uint remainBal=IERC20(_fromTokenAddr).balanceOf(address(this));
            if(remainBal>allUserDiposit[order.fromTokenAddr]){//剩余的是手续费
                TransferHelper.safeTransfer(
                    order.fromTokenAddr,
                    feeAddr,
                    remainBal.sub(allUserDiposit[order.fromTokenAddr])
                );
            }
        }else{
            reserveNum=_fromTokenNumber;
            allUserDiposit[order.fromTokenAddr]=allUserDiposit[order.fromTokenAddr].add(reserveNum);
        }
        myTradeOrderBookExt.liquidityPrice(order.fromTokenAddr,order.toTokenAddr,pairAddr,reserveA,re
        if(myTradeOrderMining!=address(0)){
            IMyTradeOrderMining(myTradeOrderMining).updateOrderMiningNumByOuter();
        }
    }
    function checkTrade(
        TokenPair storage _tokenPair,
        uint256 orderIndex,
        Order memory o,
        uint256 reserveA,
        uint256 reserveB
    ) internal {
        if(o.fromTokenNumber.mul(reserveB)>o.toTokenNumber.mul(reserveA)){//如果流动池价格低于当前价格
            uint256 cInAmount=OrderBookHelper.getInAmount(
                o.fromTokenNumber,o.toTokenNumber,reserveA,reserveB
            );//计算达到当前订单价格需要付出的币数量
            if(cInAmount>o.fromTokenNumber){
                cInAmount=o.fromTokenNumber;
            }
            uint[3] memory numArray=[0,0,_tokenPair.lastIndex[o.toTokenAddr]];
            if(numArray[2]!=0){//如果存在挂单
                Order storage bom=_tokenPair.orderMap[numArray[2]];
                Order memory bo=bom;
                if(o.fromTokenNumber.mul(bo.fromTokenNumber)>=o.toTokenNumber.mul(bo.toTokenNumber)){
                    uint256 newInAmount;
                    if(o.fromTokenNumber.mul(bo.fromTokenNumber)==o.toTokenNumber.mul(bo.toTokenNumbe
                        newInAmount=cInAmount;
                    }else{
                        newInAmount=OrderBookHelper.getInAmount(//计算对手订单价格需要付出的币数量
                        bo.toTokenNumber,bo.fromTokenNumber,reserveA,reserveB);
                    }
                    if(cInAmount>=newInAmount){//全部成交超过对手订单价格
                        uint256 tokenANum=o.fromTokenNumber.sub(newInAmount);
                        while(numArray[2]>0&&tokenANum>0){
                            uint tonum=getToNum(bo);
                            uint tonumsFee=tonum.mul(997).div(1000);
                            if(tokenANum>=tonum){//如果全部成交也不够
                                if(bo.toTokenAddr==WETH&&isForEth[bo.fromTokenAddr][numArray[2]]==2){
                                    IWETH(WETH).withdraw(tonumsFee);
                                    TransferHelper.safeTransferETH(
                                      bo.maker,
```

```
                tonumsFee
            );
        }else{
            TransferHelper.safeTransfer(
                bo.toTokenAddr,
                bo.maker,
                tonumsFee
            );
        }
        numArray[0]=numArray[0].add(tonum);//付出的
        numArray[1]=numArray[1].add(bo.remainNumber);//得到的
        bom.remainNumber=0;
        myTradeOrderBookExt.updateOrderInfo(
            bo.fromTokenAddr,
            bo.toTokenAddr,
            numArray[2],
            tonumsFee,//成交数量
            bo.remainNumber
        );
        uint newBIndex=_tokenPair.orderNextSequence[numArray[2]];//继续向上一单
        _tokenPair.orderNextSequence[numArray[2]]=0;
        numArray[2]=newBIndex;
        _tokenPair.lastIndex[o.toTokenAddr]=numArray[2];
        if(numArray[2]!=0){
            _tokenPair.orderPreSequence[numArray[2]]=0;
            bom=_tokenPair.orderMap[numArray[2]];
            bo=bom;
            if(o.fromTokenNumber.mul(bo.fromTokenNumber)>=o.toTokenNumber.mul
                newInAmount=OrderBookHelper.getInAmount(
                bo.toTokenNumber,bo.fromTokenNumber,reserveA,reserveB);//计算
                uint atemp=newInAmount.add(numArray[0]);
                if(cInAmount>=newInAmount&&o.fromTokenNumber>atemp){//继续向上
                    tokenANum=o.fromTokenNumber.sub(atemp);
                }else{
                    tokenANum=0;//停止向上遍列
                }
            }else{
                tokenANum=0;//停止向上遍列
            }
        }
    }else{
        //如果最后一条订单簿能成交够,部分成交订单簿
        uint256 atoNum=tokenANum.mul(997).div(1000);
        if(bo.toTokenAddr==WETH&&isForEth[bo.fromTokenAddr][numArray[2]]==2){
            IWETH(WETH).withdraw(atoNum);
            TransferHelper.safeTransferETH(
                bo.maker,
                atoNum
            );
        }else{
            TransferHelper.safeTransfer(
                bo.toTokenAddr,
                bo.maker,
                atoNum
            );
        }
        numArray[0]=numArray[0].add(tokenANum);
        uint256 tokenBNum=atoNum.mul(bo.fromTokenNumber) / bo.toTokenNumber;
        numArray[1]=numArray[1].add(tokenBNum);
        bom.remainNumber=bo.remainNumber.sub(tokenBNum);
        myTradeOrderBookExt.updateOrderInfo(
            bo.fromTokenAddr,
            bo.toTokenAddr,
            numArray[2],
            atoNum,//成交数量
            tokenBNum
```

```
                        );
                        tokenANum=0;//停止向上遍列
                    }
                }
            }
        }
        uint256 inAmountToliq=o.fromTokenNumber.sub(numArray[0]);
        if(cInAmount<inAmountToliq){
            inAmountToliq=cInAmount;
        }
        if(inAmountToliq>0){
            numArray[0]=numArray[0].add(inAmountToliq);
            {
                address _fromTokenAddr=o.fromTokenAddr;
                address _toTokenAddr=o.toTokenAddr;
                uint _reserveA=reserveA;
                uint _reserveB=reserveB;
                uint256 numerator = _reserveA.mul(1000);
                uint256 denominator =_reserveB.sub(1).mul(997);
                uint256 amountIn = (numerator / denominator).add(2);
                if(inAmountToliq>=amountIn){
                    uint256 amountOut=OrderBookHelper.getAmountOut(inAmountToliq,_reserveA,_reser
                    numArray[1]=numArray[1].add(amountOut);
                    address pairAddr=uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
                    TransferHelper.safeTransfer(
                        _fromTokenAddr,
                        pairAddr,
                        inAmountToliq
                    );
                    (address token0,) =OrderBookHelper.sortTokens(_fromTokenAddr, _toTokenAddr);
                    if(_fromTokenAddr == token0){
                        IUniswapV2Pair(pairAddr).swap(
                            0, amountOut, address(this), new bytes(0)
                        );
                    }else{
                        IUniswapV2Pair(pairAddr).swap(
                            amountOut, 0, address(this), new bytes(0)
                        );
                    }
                    allUserDiposit[_toTokenAddr]=allUserDiposit[_toTokenAddr].add(amountOut);
                }
            }
        }
        if(numArray[0]>0){
            if(numArray[1]>0){
                if(o.toTokenAddr==WETH&&isForEth[o.fromTokenAddr][orderIndex]==2){
                    IWETH(WETH).withdraw(numArray[1]);
                    TransferHelper.safeTransferETH(
                        o.maker,
                        numArray[1]
                    );
                }else{
                    TransferHelper.safeTransfer(
                        o.toTokenAddr,
                        o.maker,
                        numArray[1]
                    );
                }
                if (swapMining != address(0)) {
                    ISwapMining(swapMining).swap(o.maker, o.fromTokenAddr, o.toTokenAddr, numArra
                }
                allUserDiposit[o.toTokenAddr]=allUserDiposit[o.toTokenAddr].sub(numArray[1]);
            }
            myTradeOrderBookExt.updateOrderInfo(
                _tokenPair.orderMap[orderIndex].fromTokenAddr,
```

```
                    _tokenPair.orderMap[orderIndex].toTokenAddr,
                    orderIndex,
                    numArray[1],//成交数量
                    numArray[0]
                );
                _tokenPair.orderMap[orderIndex].remainNumber=
                    o.fromTokenNumber.sub(numArray[0]);

            }
        }else {
            uint[3] memory numArray=[0,0,_tokenPair.lastIndex[o.toTokenAddr]];
            uint256 tokenANum=o.fromTokenNumber;
            while(numArray[2]!=0&&tokenANum!=0){
                Order storage bo=_tokenPair.orderMap[numArray[2]];
                if(o.fromTokenNumber.mul(bo.fromTokenNumber)==o.toTokenNumber.mul(bo.toTokenNumber)){
                    tokenANum=o.fromTokenNumber.sub(numArray[0]);
                    uint256 toNum=getToNum(bo);
                    if(tokenANum>=toNum){
                        uint256 atoNum=toNum.mul(997).div(1000);
                        if(bo.toTokenAddr==WETH&&isForEth[bo.fromTokenAddr][numArray[2]]==2){
                            IWETH(WETH).withdraw(atoNum);
                            TransferHelper.safeTransferETH(
                                bo.maker,
                                atoNum
                            );
                        }else{
                            TransferHelper.safeTransfer(
                                bo.toTokenAddr,
                                bo.maker,
                                atoNum
                            );
                        }
                        numArray[0]=numArray[0].add(toNum);
                        numArray[1]=numArray[1].add(bo.remainNumber);
                        myTradeOrderBookExt.updateOrderInfo(
                            _tokenPair.orderMap[numArray[2]].fromTokenAddr,
                            _tokenPair.orderMap[numArray[2]].toTokenAddr,
                            numArray[2],
                            atoNum,//成交数量
                            bo.remainNumber
                        );
                        uint newBIndex=_tokenPair.orderNextSequence[numArray[2]];
                        _tokenPair.lastIndex[o.toTokenAddr]=newBIndex;
                        if(newBIndex!=0){
                            _tokenPair.orderNextSequence[numArray[2]]=0;
                            _tokenPair.orderPreSequence[newBIndex]=0;
                        }
                        _tokenPair.orderMap[numArray[2]].remainNumber=0;
                        numArray[2]=newBIndex;
                    }else{
                        uint256 atoNum=tokenANum.mul(997).div(1000);
                        if(bo.toTokenAddr==WETH&&isForEth[bo.fromTokenAddr][numArray[2]]==2){
                            IWETH(WETH).withdraw(atoNum);
                            TransferHelper.safeTransferETH(
                                bo.maker,
                                atoNum
                            );
                        }else{
                            TransferHelper.safeTransfer(
                                bo.toTokenAddr,
                                bo.maker,
                                atoNum
                            );
                        }

                        numArray[0]=o.fromTokenNumber;
```

```
                    uint256 tokenBNum=atoNum.mul(bo.fromTokenNumber) / bo.toTokenNumber;
                    numArray[1]=numArray[1].add(tokenBNum);
                    _tokenPair.orderMap[numArray[2]].remainNumber=
                        bo.remainNumber.sub(tokenBNum);
                    myTradeOrderBookExt.updateOrderInfo(
                        _tokenPair.orderMap[numArray[2]].fromTokenAddr,
                        _tokenPair.orderMap[numArray[2]].toTokenAddr,
                        numArray[2],
                        atoNum,//成交数量
                        tokenBNum
                    );
                    break;
                }
            }else{
                break;
            }
        }
        if(numArray[0]>0){
            if(o.toTokenAddr==WETH&&isForEth[o.fromTokenAddr][orderIndex]==2){
                IWETH(WETH).withdraw(numArray[1]);
                TransferHelper.safeTransferETH(
                    o.maker,
                    numArray[1]
                );
            }else{
                TransferHelper.safeTransfer(
                    o.toTokenAddr,
                    o.maker,
                    numArray[1]
                );
            }

            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(o.maker, o.fromTokenAddr, o.toTokenAddr, numArray[1]
            }
            allUserDiposit[o.toTokenAddr]=allUserDiposit[o.toTokenAddr].sub(numArray[1]);
            myTradeOrderBookExt.updateOrderInfo(
                _tokenPair.orderMap[orderIndex].fromTokenAddr,
                _tokenPair.orderMap[orderIndex].toTokenAddr,
               orderIndex,
                numArray[1],//成交数量
                numArray[0]
            );

            _tokenPair.orderMap[orderIndex].remainNumber=
                o.fromTokenNumber.sub(numArray[0]);
        }
     }

}

function getOrderByIndexBatch(
    address _fromTokenAddr,// 卖出token地址
    address _toTokenAddr,// 买入token地址
    uint256[] memory _orderIndexs//必须是已存在的orderIndex，否则会得不到正确结果
)public view returns(
    address[] memory makers,//挂单者
    address[] memory fromTokenAddrs,// 代币地址
    uint256[] memory fromTokenNumbers,//初始挂单量
    uint256[] memory timestamps,//初始挂单时间
    uint256[] memory remainNumbers,//当前挂单存量
    uint256[] memory toTokenNumbers,//初始意向代币目标金额
    uint256[] memory toTokenSums//已经获取的金额
){
    address pairAddr = uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
    if(pairAddr!=address(0)){
```

```
            uint256 tokenPairIndex=tokenPairIndexMap[pairAddr];
            if(tokenPairIndex!=0){
                TokenPair storage tokenPair=tokenPairArray[tokenPairIndex];
                uint256 l=_orderIndexs.length;
                makers=new  address[](l);
                fromTokenAddrs=new address[](l);
                fromTokenNumbers=new uint256[](l);
                remainNumbers=new uint256[](l);
                toTokenNumbers=new uint256[](l);
                timestamps=new uint256[](l);
                toTokenSums=new uint256[](l);
                for(uint256 i=0;i<l;i++){
                    {
                        uint256 _orderIndex=_orderIndexs[i];
                        Order memory o=tokenPair.orderMap[_orderIndex];
                        makers[i]=o.maker;
                        fromTokenAddrs[i]=o.fromTokenAddr;
                        toTokenNumbers[i]=o.toTokenNumber;
                        fromTokenNumbers[i]=o.fromTokenNumber;
                        remainNumbers[i]=o.remainNumber;
                        (timestamps[i],toTokenSums[i])=myTradeOrderBookExt.getOrderInfo(o.fromTokenAd
                    }
                }
            }
        }
    }

    function getPageOrders(// 分页获取所有订单号
        address _fromTokenAddr,// 卖出token地址
        address _toTokenAddr,// 买入token地址
        uint256 _orderStartIndex,// 订单序号点
        uint256 _records// 每次获取的个数
    )public view returns(uint256[] memory orderIndexs){
        address pairAddr = uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
        if(pairAddr!=address(0)){
            uint256 tokenPairIndex=tokenPairIndexMap[pairAddr];
            if(tokenPairIndex!=0){
                TokenPair storage tokenPair=tokenPairArray[tokenPairIndex];
                if(tokenPair.orderNextSequence[_orderStartIndex]==0){
                    uint256 ordersLastIndex=tokenPair.lastIndex[_fromTokenAddr];
                    if(tokenPair.orderPreSequence[_orderStartIndex]!=0||
                    _orderStartIndex==ordersLastIndex){
                        orderIndexs=new uint256[](1);
                        orderIndexs[0]=_orderStartIndex;
                    }
                }else{
                    orderIndexs=new uint256[](1);
                    orderIndexs[0]=_orderStartIndex;
                    if(_records!=1){
                        uint256[] memory newOrderIndexs=OrderBookHelper.joinNumber(
                            tokenPair.orderNextSequence[orderIndexs[0]],orderIndexs);
                        uint256 ll=newOrderIndexs.length;//新数组长度
                        uint256 orderNextSequence=tokenPair.orderNextSequence[newOrderIndexs[ll-1]];
                        while(orderNextSequence>0&&ll<_records){
                            newOrderIndexs=OrderBookHelper.joinNumber(orderNextSequence,newOrderIndex
                            if(ll==newOrderIndexs.length){//新数组长度没变停止
                                break;
                            }
                            ll=newOrderIndexs.length;//更新新数组长度
                            orderNextSequence=tokenPair.orderNextSequence[newOrderIndexs[ll-1]];
                        }
                        orderIndexs=newOrderIndexs;
                    }
                }
            }
        }
```

```
    }

    function getClosestOrderIndex(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _targetOrderIndex,
        uint256 _fromTokenNumber,
        uint256 _toTokenNumber,
        uint256 _depth
    )public view returns (uint256 closestOrderIndex,uint8 isEnd){
        if(_depth==0){
            return (_targetOrderIndex,0);
        }
        address pairAddr = uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
        if(pairAddr!=address(0)){
            uint256 tokenPairIndex=tokenPairIndexMap[pairAddr];
            if(tokenPairIndex!=0){
                TokenPair storage _tokenPair=tokenPairArray[tokenPairIndex];
                if(_tokenPair.orderNextSequence[_targetOrderIndex]==0
                    &&_tokenPair.orderPreSequence[_targetOrderIndex]==0){
                    _targetOrderIndex=_tokenPair.lastIndex[_fromTokenAddr];
                }
                if(_targetOrderIndex!=0){
                    Order memory o=_tokenPair.orderMap[_targetOrderIndex];
                    if(o.fromTokenAddr!=_fromTokenAddr){
                        _targetOrderIndex=_tokenPair.lastIndex[_fromTokenAddr];
                    }
                    if(_targetOrderIndex!=0){
                        if(_fromTokenNumber.mul(_tokenPair.orderMap[_targetOrderIndex].toTokenNumber)
                            _tokenPair.orderMap[_targetOrderIndex].fromTokenNumber.mul(_toTokenNumber
                            uint256 orderNextSequence=_tokenPair.orderNextSequence[_targetOrderIndex]
                            if(orderNextSequence==0){
                               return (_targetOrderIndex,1);
                            }else{
                                if(_fromTokenNumber.mul(_tokenPair.orderMap[orderNextSequence].toToke
                                    _tokenPair.orderMap[orderNextSequence].fromTokenNumber.mul(_toTok
                                    return getClosestOrderIndex(
                                        _fromTokenAddr,
                                        _toTokenAddr,
                                        orderNextSequence,
                                        _fromTokenNumber,
                                        _toTokenNumber,
                                        _depth.sub(1)
                                    );
                                }else{
                                    return (_targetOrderIndex,1);
                                }
                            }
                        }else{
                            uint256 orderPreIndex=_tokenPair.orderPreSequence[_targetOrderIndex];
                            if(orderPreIndex==0){
                                return (_targetOrderIndex,1);
                            }else{
                                if(_fromTokenNumber.mul(_tokenPair.orderMap[orderPreIndex].toTokenNum
                                    _tokenPair.orderMap[orderPreIndex].fromTokenNumber.mul(_toTokenNu
                                     return getClosestOrderIndex(
                                        _fromTokenAddr,
                                        _toTokenAddr,
                                        orderPreIndex,
                                        _fromTokenNumber,
                                        _toTokenNumber,
                                        _depth.sub(1)
                                    );
                                }else{
                                  return (_targetOrderIndex,1);
                                }
```

```
                            }
                        }
                    }
                }
            }
        }
    }
    function getLastOrderIndex(
        address _fromTokenAddr,
        address _toTokenAddr
    )public view returns (uint256 lastOrderIndex){
        address pairAddr = uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
        if(pairAddr!=address(0)){
            uint256 tokenPairIndex=tokenPairIndexMap[pairAddr];
            if(tokenPairIndex!=0){
                lastOrderIndex=tokenPairArray[tokenPairIndex].lastIndex[_fromTokenAddr];
            }
        }
    }
    function getNextOrderIndex(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _orderIndex
    )public view returns  (uint256 nextOrderIndex){
        address pairAddr = uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
        if(pairAddr!=address(0)){
            uint256 tokenPairIndex=tokenPairIndexMap[pairAddr];
            if(tokenPairIndex!=0){
                if(tokenPairArray[tokenPairIndex].orderMap[_orderIndex].fromTokenAddr==_fromTokenAddr
                    nextOrderIndex=tokenPairArray[tokenPairIndex].orderNextSequence[_orderIndex];
                }
            }
        }
    }
    function getPreOrderIndex(
        address _fromTokenAddr,
        address _toTokenAddr,
        uint256 _orderIndex
    )public view returns (uint256 preOrderIndex){
        address pairAddr = uniswapV2Factory.getPair(_fromTokenAddr, _toTokenAddr);
        uint256 tokenPairIndex=tokenPairIndexMap[pairAddr];
        if(tokenPairIndex!=0){
            if(tokenPairArray[tokenPairIndex].orderMap[_orderIndex].fromTokenAddr==_fromTokenAddr){
                preOrderIndex=tokenPairArray[tokenPairIndex].orderPreSequence[_orderIndex];
            }
        }
    }
    function getToNum(Order memory bo) internal pure returns (uint256 _to){
        _to=bo.toTokenNumber.mul(
            bo.remainNumber
        ).div(bo.fromTokenNumber).mul(1000).div(997);
    }

    // fetches and sorts the reserves for a pair
    function getReserves(address tokenA, address tokenB) internal view returns (uint reserveA, uint r
        (address token0,) = OrderBookHelper.sortTokens(tokenA, tokenB);
        (uint reserve0, uint reserve1,) = IUniswapV2Pair(uniswapV2Factory.getPair(tokenA, tokenB)).ge
        (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
    }

}
interface IFlashLoanService {
     function check(address from, uint256 value) external returns (bool);
}

contract FlashLoan is ReentrancyGuard,Ownable{
```

```
    address public loanServiceAddr;
    constructor(
        address _owner
    ){
       _transferOwnership(_owner);
    }
    function setLoanServiceAddr(address _loanServiceAddr)public onlyOwner {
        loanServiceAddr=_loanServiceAddr;
    }
    function loan(
        address from,
        address token,
        uint value,
        address contractAddr,
        bytes memory msgdata
    ) public nonReentrant returns(bool){
       uint fromBal=IERC20(token).balanceOf(from);
       require(fromBal>=value,"insufficient balance");
       TransferHelper.safeTransferFrom(token,from,contractAddr,value);
       (bool success, bytes memory data) =contractAddr.call(msgdata);
       require(success && (data.length == 0 || abi.decode(data, (bool))), 'loan failed');
       require(IFlashLoanService(loanServiceAddr).check(from,fromBal));
       require(IERC20(token).balanceOf(from)>=fromBal,"error balance:loan failed");
       return true;
    }
}
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.8.0;
contract Ownable {
    address private _owner;

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

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor () {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
```

```
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     * @notice Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0));
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
contract ReentrancyGuard {
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;
    uint256 private _status;
    constructor () {
        _status = _NOT_ENTERED;
    }
    modifier nonReentrant() {
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
        _status = _ENTERED;
        _;
        _status = _NOT_ENTERED;
    }
}
library SafeMath {
    /**
     * @dev Multiplies two unsigned integers, reverts on 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-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
     * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
```

```
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Adds two unsigned integers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}
/**
 * @title ERC20 interface
 * @dev see https://eips.ethereum.org/EIPS/eip-20
 */
interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns (uint256);

}
interface IUniswapV2Factory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}
interface IUniswapV2Pair {
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
    function token0() external view returns (address);
    function token1() external view returns (address);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
}
// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
```

```
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: MyTradeOr
    }

    function safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'TransferHelper: MyTradeOrderBook ETH_TRANSFER_FAILED');
    }
}
/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 * ``
 * contract Example {
 *     // Add the library methods
 *     using EnumerableSet for EnumerableSet.AddressSet;
 *
 *     // Declare a set state variable
 *     EnumerableSet.AddressSet private mySet;
 * }
 * ``
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 */
library EnumerableSet {
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;

        // Position of the value in the `values` array, plus 1 because index 0
        // means a value is not in the set.
        mapping (bytes32 => uint256) _indexes;
    }

    /**
     * @dev Add a value to a set. O(1).
```

```
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function _add(Set storage set, bytes32 value) private returns (bool) {
        if (!_contains(set, value)) {
            set._values.push(value);
            // The value is stored at length-1, but we add 1 to all indexes
            // and use 0 as a sentinel value
            set._indexes[value] = set._values.length;
            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) { // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the element to delete wit
            // the array, and then remove the last element (sometimes called as 'swap and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            // When the value to delete is the last one, the swap operation is unnecessary. However,
            // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' stateme

            bytes32 lastvalue = set._values[lastIndex];

            // Move the last value to the index where the value to delete is
            set._values[toDeleteIndex] = lastvalue;
            // Update the index for the moved value
            set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];

            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
```

```
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        require(set._values.length > index, "EnumerableSet: index out of bounds");
        return set._values[index];
    }

    // Bytes32Set

    struct Bytes32Set {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _add(set._inner, value);
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _remove(set._inner, value);
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) {
        return _contains(set._inner, value);
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
```

```
     *
     * - `index` must be strictly less than {length}.
     */
    function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
        return _at(set._inner, index);
    }

    // AddressSet

    struct AddressSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(AddressSet storage set, uint256 index) internal view returns (address) {
        return address(uint160(uint256(_at(set._inner, index))));
    }


    // UintSet

    struct UintSet {
        Set _inner;
```

```
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
}
interface IOracle {
    function update(address tokenA, address tokenB) external;
    function consult(address tokenIn, uint amountIn, address tokenOut) external view returns (uint am
}

contract MyTradeSwapMining is Ownable,ReentrancyGuard{
     using SafeMath for uint256;
    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _whitelist;

    // tokens created per block
    uint256 public tokenPerBlock;
    // The block number when mining starts.
    uint256 public startBlock;

    // Total allocation points
```

```
    uint256 public totalAllocPoint = 0;
    IOracle public oracle;
    // orderbook address
    address public orderbook;
    // factory address
    IUniswapV2Factory public factory;
    // mytrade token address
    address public myt;
    // Calculate price based on HUSD
    address public targetToken;
    // pair corresponding pid
    mapping(address => uint256) public pairOfPid;

    constructor(
        address _myt,
        IUniswapV2Factory _factory,
        IOracle _oracle,
        address _orderbook,
        address _targetToken,
        uint256 _tokenPerBlock,
        uint256 _startBlock
    ){
        myt = _myt;
        factory = _factory;
        oracle = _oracle;
        orderbook = _orderbook;
        targetToken = _targetToken;
        tokenPerBlock = _tokenPerBlock;
        startBlock = _startBlock;
    }

    struct UserInfo {
        uint256 quantity;       // How many LP tokens the user has provided
        uint256 blockNumber;    // Last transaction block
    }

    struct PoolInfo {
        address pair;           // Trading pairs that can be mined
        uint256 quantity;       // Current amount of LPs
        uint256 totalQuantity;  // All quantity
        uint256 allocPoint;     // How many allocation points assigned to this pool
        uint256 allocTokenAmount; // How many tokens
        uint256 lastRewardBlock;// Last transaction block
    }

    PoolInfo[] public poolInfo;
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;


    function poolLength() public view returns (uint256) {
        return poolInfo.length;
    }


    function addPair(uint256 _allocPoint, address _pair, bool _withUpdate) public onlyOwner {
        require(_pair != address(0), "_pair is the zero address");
        if (_withUpdate) {
            massMintPools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
            pair : _pair,
            quantity : 0,
            totalQuantity : 0,
            allocPoint : _allocPoint,
```

```
            allocTokenAmount : 0,
            lastRewardBlock : lastRewardBlock
        }));
        pairOfPid[_pair] = poolLength() - 1;
    }

    // Update the allocPoint of the pool
    function setPair(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massMintPools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // Set the number of token produced by each block
    function setTokenPerBlock(uint256 _newPerBlock) public onlyOwner {
        massMintPools();
        tokenPerBlock = _newPerBlock;
    }

    // Only tokens in the whitelist can be mined token
    function addWhitelist(address _addToken) public onlyOwner returns (bool) {
        require(_addToken != address(0), "SwapMining: token is the zero address");
        return EnumerableSet.add(_whitelist, _addToken);
    }

    function delWhitelist(address _delToken) public onlyOwner returns (bool) {
        require(_delToken != address(0), "SwapMining: token is the zero address");
        return EnumerableSet.remove(_whitelist, _delToken);
    }

    function getWhitelistLength() public view returns (uint256) {
        return EnumerableSet.length(_whitelist);
    }

    function isWhitelist(address _token) public view returns (bool) {
        return EnumerableSet.contains(_whitelist, _token);
    }

    function getWhitelist(uint256 _index) public view returns (address){
        require(_index <= getWhitelistLength() - 1, "SwapMining: index out of bounds");
        return EnumerableSet.at(_whitelist, _index);
    }


    function setOrderbook(address newOrderbook) public onlyOwner {
        require(newOrderbook != address(0), "SwapMining: new orderbook is the zero address");
        orderbook = newOrderbook;
    }

    function setOracle(IOracle _oracle) public onlyOwner {
        require(address(_oracle) != address(0), "SwapMining: new oracle is the zero address");
        oracle = _oracle;
    }


    // Update all pools Called when updating allocPoint and setting new blocks
    function massMintPools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            mint(pid);
        }
    }
// Rewards for the current block
    function getTokenReward(uint256 _lastRewardBlock) public view returns (uint256) {
```

```
        require(_lastRewardBlock <= block.number, "SwapMining: must little than the current block num
        return (block.number.sub(_lastRewardBlock)).mul(tokenPerBlock);
    }
    address public totalPoolAddr;
    function setTotalPoolAddr(
        address _totalPoolAddr
    ) onlyOwner public returns(bool) {
        totalPoolAddr=_totalPoolAddr;
        return true;
    }
    function mint(uint256 _pid) public returns (bool) {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return false;
        }
        uint256 blockReward = getTokenReward(pool.lastRewardBlock);
        if (blockReward <= 0) {
            return false;
        }
        // Calculate the rewards obtained by the pool based on the allocPoint
        uint256 tokenReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
        TransferHelper.safeTransferFrom(
                address(myt),
                totalPoolAddr,
                address(this),
                tokenReward
            );
        // Increase the number of tokens in the current pool
        pool.allocTokenAmount = pool.allocTokenAmount.add(tokenReward);
        pool.lastRewardBlock = block.number;
        return true;
    }

    // swapMining only router
    function swap(address account, address input, address output, uint256 amount) public onlyOrderboo
        require(account != address(0), "SwapMining: taker swap account is the zero address");
        require(input != address(0), "SwapMining: taker swap input is the zero address");
        require(output != address(0), "SwapMining: taker swap output is the zero address");

        if (poolLength() <= 0) {
            return false;
        }

        if (!isWhitelist(input) || !isWhitelist(output)) {
            return false;
        }

        address pair = IUniswapV2Factory(factory).getPair(input, output);

        PoolInfo storage pool = poolInfo[pairOfPid[pair]];
        // If it does not exist or the allocPoint is 0 then return
        if (pool.pair != pair || pool.allocPoint <= 0) {
            return false;
        }

        uint256 quantity = getQuantity(output, amount, targetToken);
        if (quantity <= 0) {
            return false;
        }

        mint(pairOfPid[pair]);

        pool.quantity = pool.quantity.add(quantity);
        pool.totalQuantity = pool.totalQuantity.add(quantity);
        UserInfo storage user = userInfo[pairOfPid[pair]][account];
        user.quantity = user.quantity.add(quantity);
```

```
        user.blockNumber = block.number;
        return true;
    }

    // The user withdraws all the transaction rewards of the pool
    function takerWithdraw() public {
        uint256 userSub;
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            PoolInfo storage pool = poolInfo[pid];
            UserInfo storage user = userInfo[pid][msg.sender];
            if (user.quantity > 0) {
                mint(pid);
                // The reward held by the user in this pool
                uint256 userReward = pool.allocTokenAmount.mul(user.quantity).div(pool.quantity);
                pool.quantity = pool.quantity.sub(user.quantity);
                pool.allocTokenAmount = pool.allocTokenAmount.sub(userReward);
                user.quantity = 0;
                user.blockNumber = block.number;
                userSub = userSub.add(userReward);
            }
        }
        if (userSub <= 0) {
            return;
        }
        TransferHelper.safeTransfer(
                myt,
                msg.sender,
                userSub
            );
    }

    // Get rewards from users in the current pool
    function getUserReward(uint256 _pid) public view returns (uint256, uint256){
        require(_pid <= poolInfo.length - 1, "SwapMining: Not find this pool");
        uint256 userSub;
        PoolInfo memory pool = poolInfo[_pid];
        UserInfo memory user = userInfo[_pid][msg.sender];
        if (user.quantity > 0) {
            uint256 blockReward = getTokenReward(pool.lastRewardBlock);
            uint256 tokenReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
            userSub = userSub.add((pool.allocTokenAmount.add(tokenReward)).mul(user.quantity).div(poo
        }
        //token available to users, User transaction amount
        return (userSub, user.quantity);
    }

    // Get details of the pool
    function getPoolInfo(uint256 _pid) public view returns (address, address, uint256, uint256, uint2
        require(_pid <= poolInfo.length - 1, "SwapMining: Not find this pool");
        PoolInfo memory pool = poolInfo[_pid];
        address token0 = IUniswapV2Pair(pool.pair).token0();
        address token1 = IUniswapV2Pair(pool.pair).token1();
        uint256 tokenAmount = pool.allocTokenAmount;
        uint256 blockReward = getTokenReward(pool.lastRewardBlock);
        uint256 tokenReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
        tokenAmount = tokenAmount.add(tokenReward);
        //token0,token1,Pool remaining reward,Total /Current transaction volume of the pool
        return (token0, token1, tokenAmount, pool.totalQuantity, pool.quantity, pool.allocPoint);
    }

    modifier onlyOrderbook() {
        require(msg.sender == orderbook, "SwapMining: caller is not the router");
        _;
    }
```

```
    function getQuantity(address outputToken, uint256 outputAmount, address anchorToken) public view
        uint256 quantity = 0;
        if (outputToken == anchorToken) {
            quantity = outputAmount;
        } else if (IUniswapV2Factory(factory).getPair(outputToken, anchorToken) != address(0)) {
            quantity = IOracle(oracle).consult(outputToken, outputAmount, anchorToken);
        } else {
            uint256 length = getWhitelistLength();
            for (uint256 index = 0; index < length; index++) {
                address intermediate = getWhitelist(index);
                if (IUniswapV2Factory(factory).getPair(outputToken, intermediate) != address(0) &&
                    IUniswapV2Factory(factory).getPair(intermediate, anchorToken) != address(0)) {
                    uint256 interQuantity = IOracle(oracle).consult(outputToken, outputAmount, interm
                    quantity = IOracle(oracle).consult(intermediate, interQuantity, anchorToken);
                    break;
                }
            }
        }
        return quantity;
    }

}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing

conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

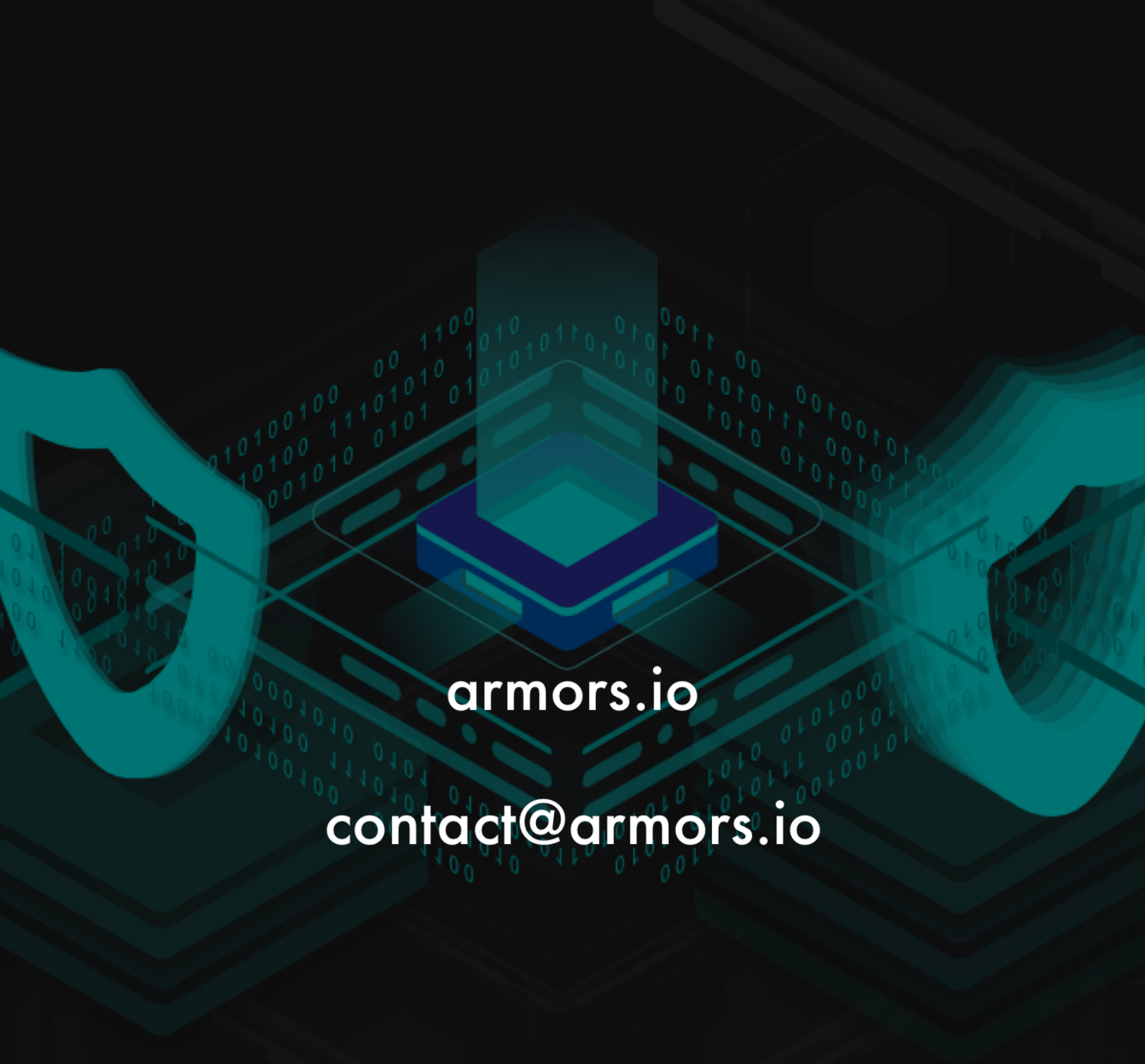
- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.



armors.io

contact@armors.io