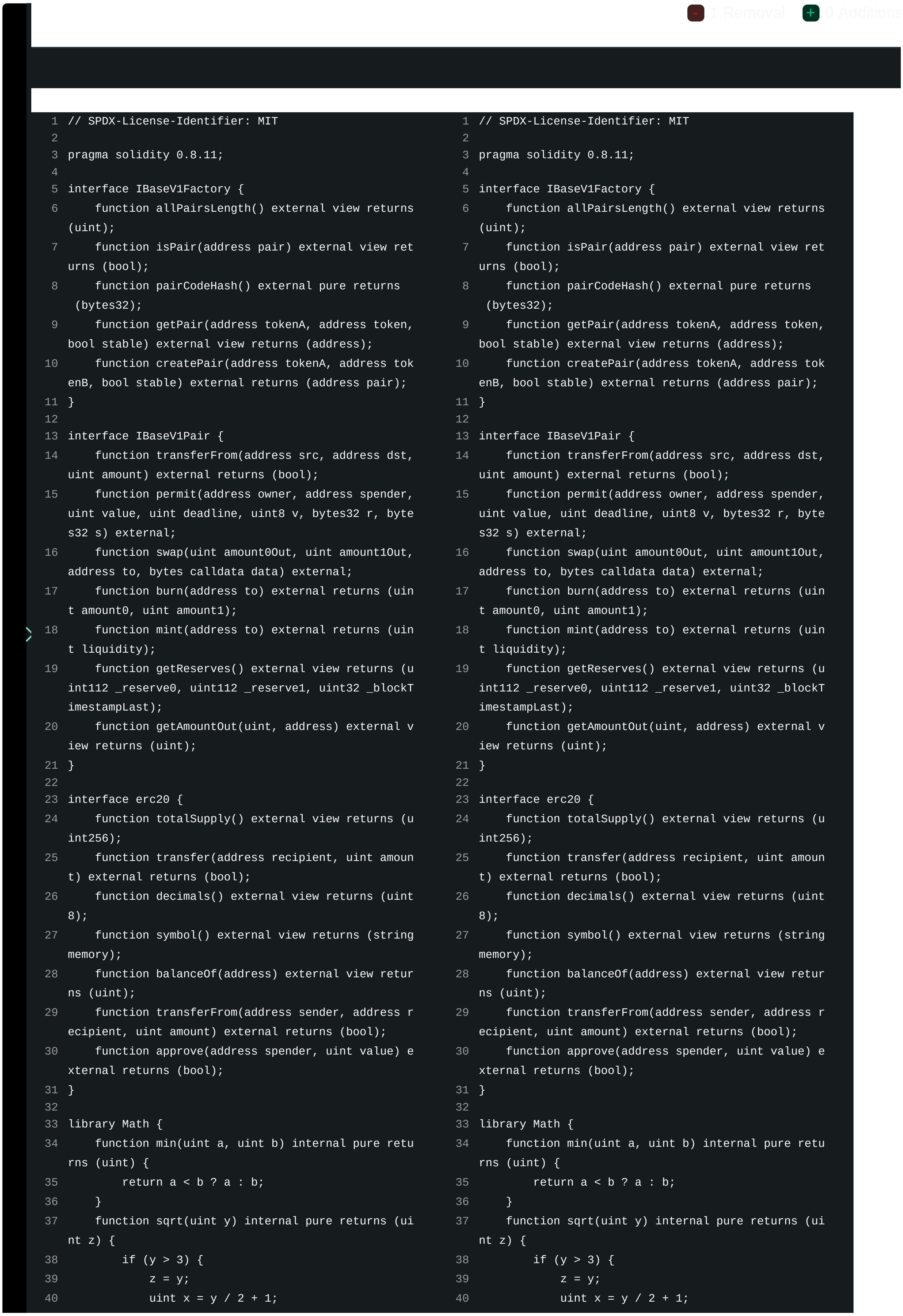
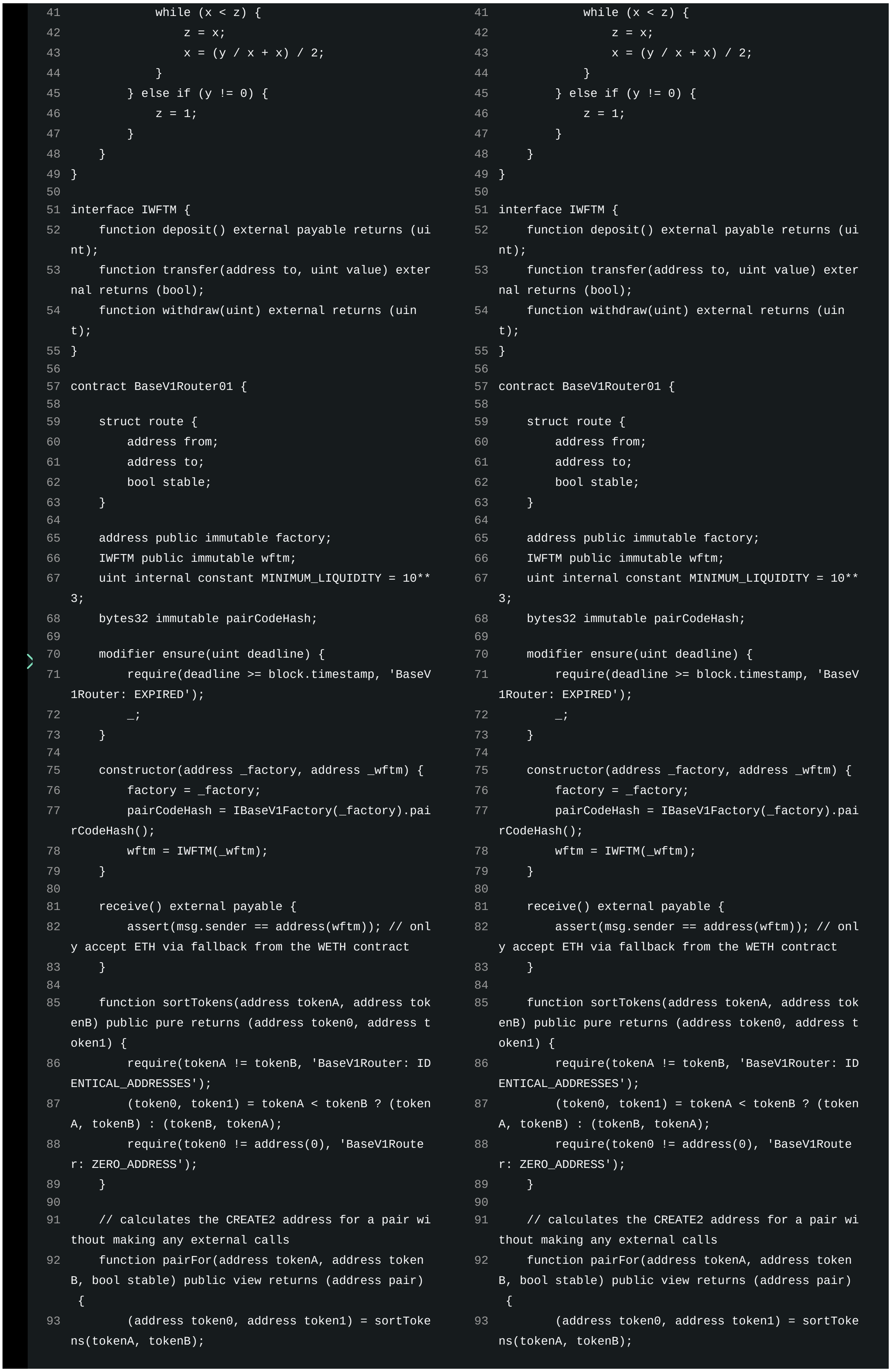


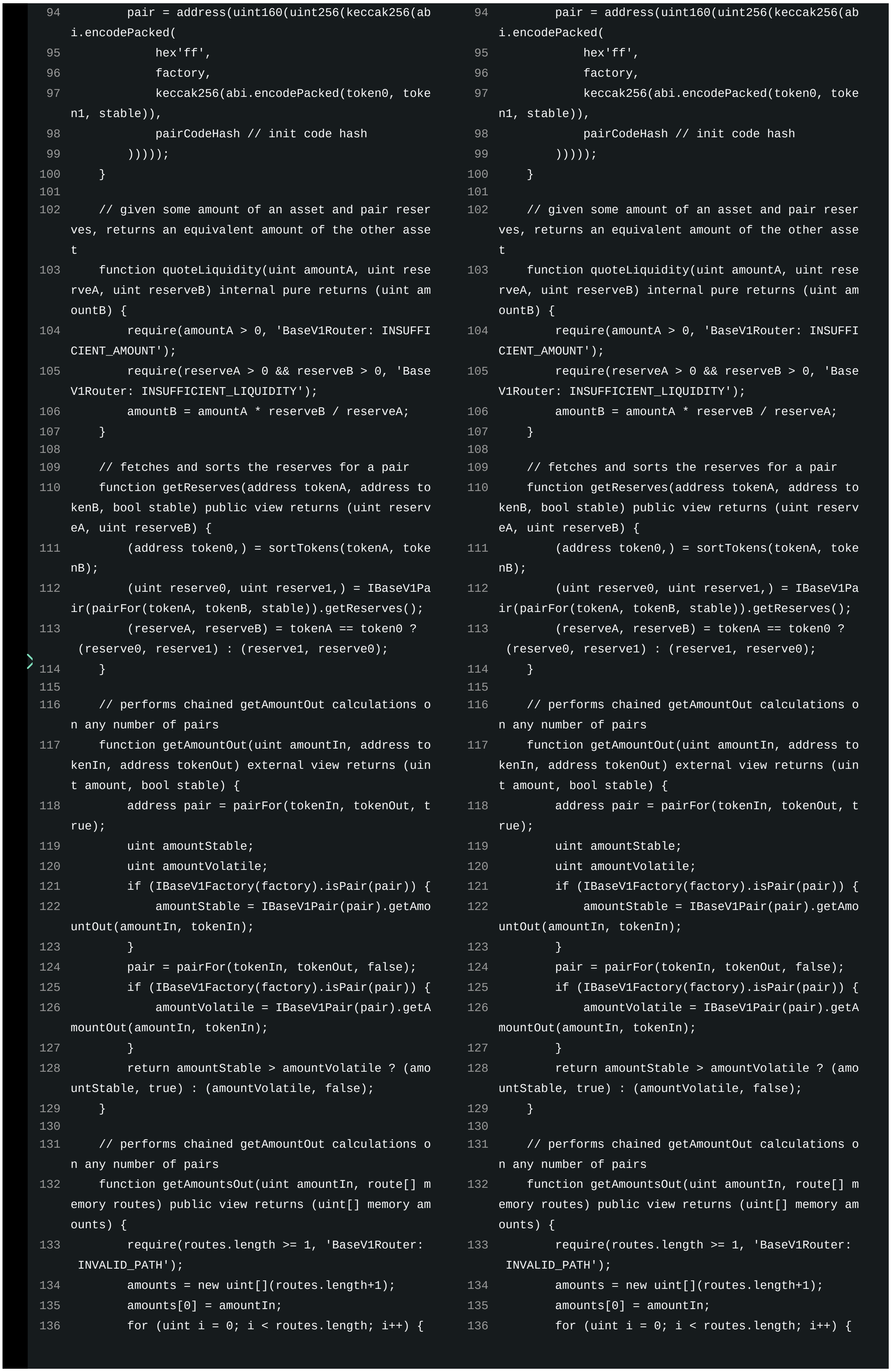
```solidity
// SPDX-License-Identifier: MIT

pragma solidity 0.8.11;

interface IBaseV1Factory {
    function allPairsLength() external view returns (uint);
    function isPair(address pair) external view returns (bool);
    function pairCodeHash() external pure returns (bytes32);
    function getPair(address tokenA, address token, bool stable) external view returns (address);
    function createPair(address tokenA, address tokenB, bool stable) external returns (address pair);
}

interface IBaseV1Pair {
    function transferFrom(address src, address dst, uint amount) external returns (bool);
    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function burn(address to) external returns (uint amount0, uint amount1);
    function mint(address to) external returns (uint liquidity);
    function getReserves() external view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast);
    function getAmountOut(uint, address) external view returns (uint);
}

interface erc20 {
    function totalSupply() external view returns (uint256);
    function transfer(address recipient, uint amount) external returns (bool);
    function decimals() external view returns (uint8);
    function symbol() external view returns (string memory);
    function balanceOf(address) external view returns (uint);
    function transferFrom(address sender, address recipient, uint amount) external returns (bool);
    function approve(address spender, uint value) external returns (bool);
}

library Math {
    function min(uint a, uint b) internal pure returns (uint) {
        return a < b ? a : b;
    }
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
```

```solidity
// SPDX-License-Identifier: MIT

pragma solidity 0.8.11;

interface IBaseV1Factory {
    function allPairsLength() external view returns (uint);
    function isPair(address pair) external view returns (bool);
    function pairCodeHash() external pure returns (bytes32);
    function getPair(address tokenA, address token, bool stable) external view returns (address);
    function createPair(address tokenA, address tokenB, bool stable) external returns (address pair);
}

interface IBaseV1Pair {
    function transferFrom(address src, address dst, uint amount) external returns (bool);
    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function burn(address to) external returns (uint amount0, uint amount1);
    function mint(address to) external returns (uint liquidity);
    function getReserves() external view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast);
    function getAmountOut(uint, address) external view returns (uint);
}

interface erc20 {
    function totalSupply() external view returns (uint256);
    function transfer(address recipient, uint amount) external returns (bool);
    function decimals() external view returns (uint8);
    function symbol() external view returns (string memory);
    function balanceOf(address) external view returns (uint);
    function transferFrom(address sender, address recipient, uint amount) external returns (bool);
    function approve(address spender, uint value) external returns (bool);
}

library Math {
    function min(uint a, uint b) internal pure returns (uint) {
        return a < b ? a : b;
    }
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
```



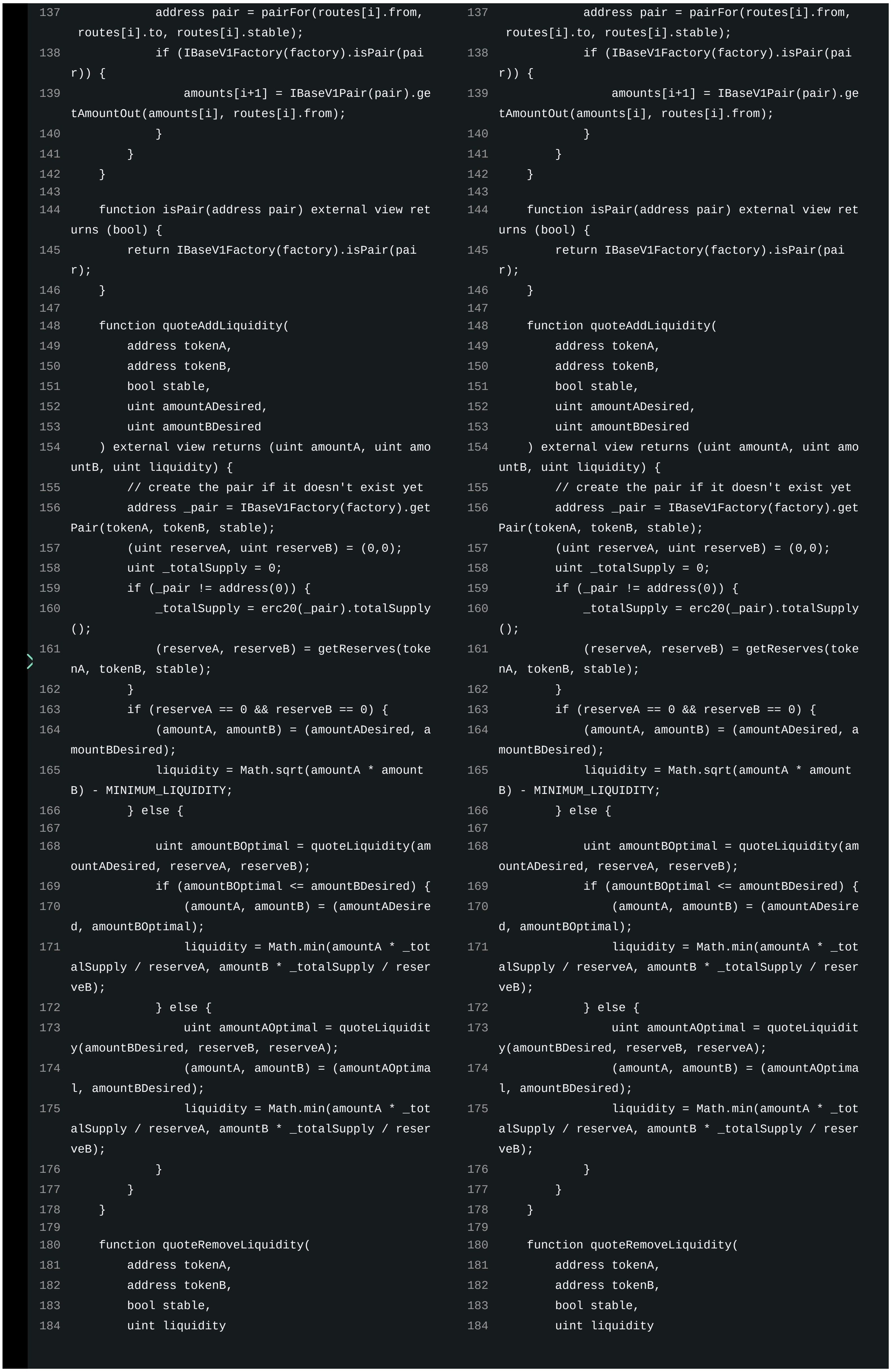
```
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}

interface IWFTM {
    function deposit() external payable returns (uint);
    function transfer(address to, uint value) external returns (bool);
    function withdraw(uint) external returns (uint);
}

contract BaseV1Router01 {

    struct route {
        address from;
        address to;
        bool stable;
    }

    address public immutable factory;
    IWFTM public immutable wftm;
    uint internal constant MINIMUM_LIQUIDITY = 10**3;
    bytes32 immutable pairCodeHash;

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'BaseV1Router: EXPIRED');
        _;
    }

    constructor(address _factory, address _wftm) {
        factory = _factory;
        pairCodeHash = IBaseV1Factory(_factory).pairCodeHash();
        wftm = IWFTM(_wftm);
    }

    receive() external payable {
        assert(msg.sender == address(wftm)); // only accept ETH via fallback from the WETH contract
    }

    function sortTokens(address tokenA, address tokenB) public pure returns (address token0, address token1) {
        require(tokenA != tokenB, 'BaseV1Router: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'BaseV1Router: ZERO_ADDRESS');
    }

    // calculates the CREATE2 address for a pair without making any external calls
    function pairFor(address tokenA, address tokenB, bool stable) public view returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
```

```
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}

interface IWFTM {
    function deposit() external payable returns (uint);
    function transfer(address to, uint value) external returns (bool);
    function withdraw(uint) external returns (uint);
}

contract BaseV1Router01 {

    struct route {
        address from;
        address to;
        bool stable;
    }

    address public immutable factory;
    IWFTM public immutable wftm;
    uint internal constant MINIMUM_LIQUIDITY = 10**3;
    bytes32 immutable pairCodeHash;

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'BaseV1Router: EXPIRED');
        _;
    }

    constructor(address _factory, address _wftm) {
        factory = _factory;
        pairCodeHash = IBaseV1Factory(_factory).pairCodeHash();
        wftm = IWFTM(_wftm);
    }

    receive() external payable {
        assert(msg.sender == address(wftm)); // only accept ETH via fallback from the WETH contract
    }

    function sortTokens(address tokenA, address tokenB) public pure returns (address token0, address token1) {
        require(tokenA != tokenB, 'BaseV1Router: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'BaseV1Router: ZERO_ADDRESS');
    }

    // calculates the CREATE2 address for a pair without making any external calls
    function pairFor(address tokenA, address tokenB, bool stable) public view returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
```



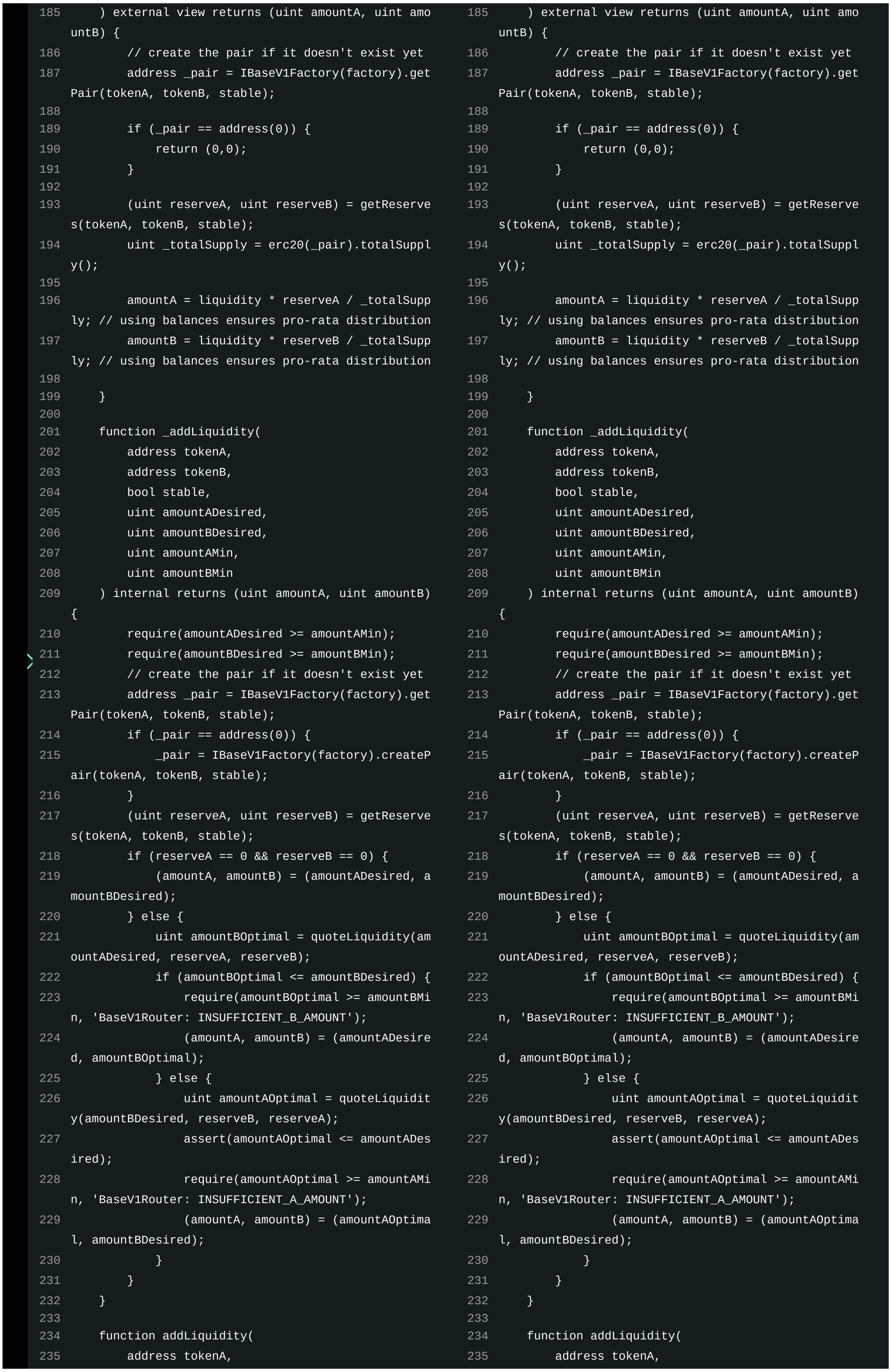
```
        pair = address(uint160(uint256(keccak256(abi.encodePacked(
            hex'ff',
            factory,
            keccak256(abi.encodePacked(token0, token1, stable)),
            pairCodeHash // init code hash
        )))));
    }

    // given some amount of an asset and pair reserves, returns an equivalent amount of the other asset
    function quoteLiquidity(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB) {
        require(amountA > 0, 'BaseV1Router: INSUFFICIENT_AMOUNT');
        require(reserveA > 0 && reserveB > 0, 'BaseV1Router: INSUFFICIENT_LIQUIDITY');
        amountB = amountA * reserveB / reserveA;
    }

    // fetches and sorts the reserves for a pair
    function getReserves(address tokenA, address tokenB, bool stable) public view returns (uint reserveA, uint reserveB) {
        (address token0,) = sortTokens(tokenA, tokenB);
        (uint reserve0, uint reserve1,) = IBaseV1Pair(pairFor(tokenA, tokenB, stable)).getReserves();
        (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
    }

    // performs chained getAmountOut calculations on any number of pairs
    function getAmountOut(uint amountIn, address tokenIn, address tokenOut) external view returns (uint amount, bool stable) {
        address pair = pairFor(tokenIn, tokenOut, true);
        uint amountStable;
        uint amountVolatile;
        if (IBaseV1Factory(factory).isPair(pair)) {
            amountStable = IBaseV1Pair(pair).getAmountOut(amountIn, tokenIn);
        }
        pair = pairFor(tokenIn, tokenOut, false);
        if (IBaseV1Factory(factory).isPair(pair)) {
            amountVolatile = IBaseV1Pair(pair).getAmountOut(amountIn, tokenIn);
        }
        return amountStable > amountVolatile ? (amountStable, true) : (amountVolatile, false);
    }

    // performs chained getAmountOut calculations on any number of pairs
    function getAmountsOut(uint amountIn, route[] memory routes) public view returns (uint[] memory amounts) {
        require(routes.length >= 1, 'BaseV1Router: INVALID_PATH');
        amounts = new uint[](routes.length+1);
        amounts[0] = amountIn;
        for (uint i = 0; i < routes.length; i++) {
```

```
        pair = address(uint160(uint256(keccak256(abi.encodePacked(
            hex'ff',
            factory,
            keccak256(abi.encodePacked(token0, token1, stable)),
            pairCodeHash // init code hash
        )))));
    }

    // given some amount of an asset and pair reserves, returns an equivalent amount of the other asset
    function quoteLiquidity(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB) {
        require(amountA > 0, 'BaseV1Router: INSUFFICIENT_AMOUNT');
        require(reserveA > 0 && reserveB > 0, 'BaseV1Router: INSUFFICIENT_LIQUIDITY');
        amountB = amountA * reserveB / reserveA;
    }

    // fetches and sorts the reserves for a pair
    function getReserves(address tokenA, address tokenB, bool stable) public view returns (uint reserveA, uint reserveB) {
        (address token0,) = sortTokens(tokenA, tokenB);
        (uint reserve0, uint reserve1,) = IBaseV1Pair(pairFor(tokenA, tokenB, stable)).getReserves();
        (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
    }

    // performs chained getAmountOut calculations on any number of pairs
    function getAmountOut(uint amountIn, address tokenIn, address tokenOut) external view returns (uint amount, bool stable) {
        address pair = pairFor(tokenIn, tokenOut, true);
        uint amountStable;
        uint amountVolatile;
        if (IBaseV1Factory(factory).isPair(pair)) {
            amountStable = IBaseV1Pair(pair).getAmountOut(amountIn, tokenIn);
        }
        pair = pairFor(tokenIn, tokenOut, false);
        if (IBaseV1Factory(factory).isPair(pair)) {
            amountVolatile = IBaseV1Pair(pair).getAmountOut(amountIn, tokenIn);
        }
        return amountStable > amountVolatile ? (amountStable, true) : (amountVolatile, false);
    }

    // performs chained getAmountOut calculations on any number of pairs
    function getAmountsOut(uint amountIn, route[] memory routes) public view returns (uint[] memory amounts) {
        require(routes.length >= 1, 'BaseV1Router: INVALID_PATH');
        amounts = new uint[](routes.length+1);
        amounts[0] = amountIn;
        for (uint i = 0; i < routes.length; i++) {
```



```
            address pair = pairFor(routes[i].from, routes[i].to, routes[i].stable);
            if (IBaseV1Factory(factory).isPair(pair)) {
                amounts[i+1] = IBaseV1Pair(pair).getAmountOut(amounts[i], routes[i].from);
            }
        }
    }

    function isPair(address pair) external view returns (bool) {
        return IBaseV1Factory(factory).isPair(pair);
    }

    function quoteAddLiquidity(
        address tokenA,
        address tokenB,
        bool stable,
        uint amountADesired,
        uint amountBDesired
    ) external view returns (uint amountA, uint amountB, uint liquidity) {
        // create the pair if it doesn't exist yet
        address _pair = IBaseV1Factory(factory).getPair(tokenA, tokenB, stable);
        (uint reserveA, uint reserveB) = (0,0);
        uint _totalSupply = 0;
        if (_pair != address(0)) {
            _totalSupply = erc20(_pair).totalSupply();
            (reserveA, reserveB) = getReserves(tokenA, tokenB, stable);
        }
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
            liquidity = Math.sqrt(amountA * amountB) - MINIMUM_LIQUIDITY;
        } else {

            uint amountBOptimal = quoteLiquidity(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                (amountA, amountB) = (amountADesired, amountBOptimal);
                liquidity = Math.min(amountA * _totalSupply / reserveA, amountB * _totalSupply / reserveB);
            } else {
                uint amountAOptimal = quoteLiquidity(amountBDesired, reserveB, reserveA);
                (amountA, amountB) = (amountAOptimal, amountBDesired);
                liquidity = Math.min(amountA * _totalSupply / reserveA, amountB * _totalSupply / reserveB);
            }
        }
    }

    function quoteRemoveLiquidity(
        address tokenA,
        address tokenB,
        bool stable,
        uint liquidity
```

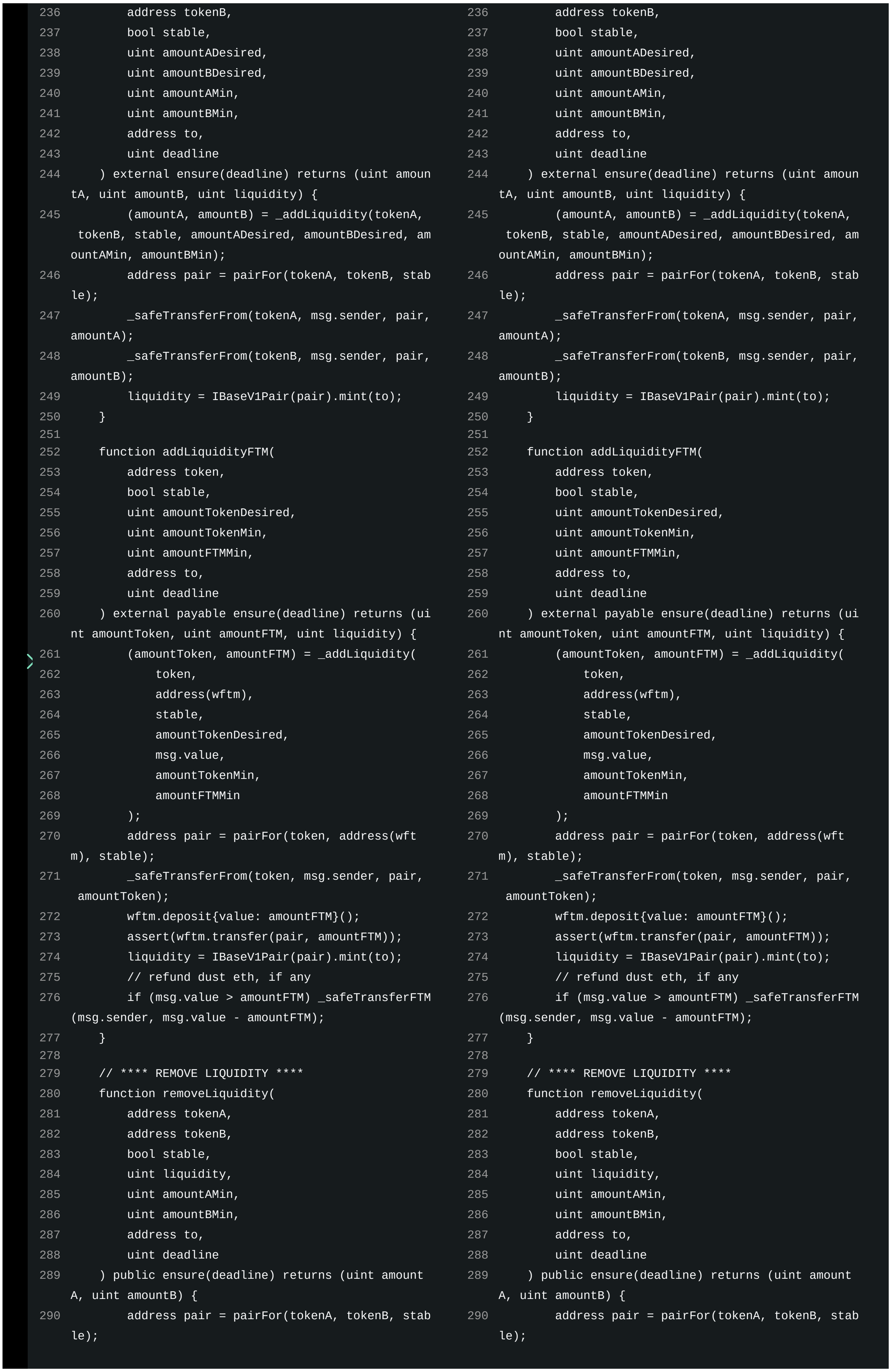
```
            address pair = pairFor(routes[i].from, routes[i].to, routes[i].stable);
            if (IBaseV1Factory(factory).isPair(pair)) {
                amounts[i+1] = IBaseV1Pair(pair).getAmountOut(amounts[i], routes[i].from);
            }
        }
    }

    function isPair(address pair) external view returns (bool) {
        return IBaseV1Factory(factory).isPair(pair);
    }

    function quoteAddLiquidity(
        address tokenA,
        address tokenB,
        bool stable,
        uint amountADesired,
        uint amountBDesired
    ) external view returns (uint amountA, uint amountB, uint liquidity) {
        // create the pair if it doesn't exist yet
        address _pair = IBaseV1Factory(factory).getPair(tokenA, tokenB, stable);
        (uint reserveA, uint reserveB) = (0,0);
        uint _totalSupply = 0;
        if (_pair != address(0)) {
            _totalSupply = erc20(_pair).totalSupply();
            (reserveA, reserveB) = getReserves(tokenA, tokenB, stable);
        }
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
            liquidity = Math.sqrt(amountA * amountB) - MINIMUM_LIQUIDITY;
        } else {

            uint amountBOptimal = quoteLiquidity(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                (amountA, amountB) = (amountADesired, amountBOptimal);
                liquidity = Math.min(amountA * _totalSupply / reserveA, amountB * _totalSupply / reserveB);
            } else {
                uint amountAOptimal = quoteLiquidity(amountBDesired, reserveB, reserveA);
                (amountA, amountB) = (amountAOptimal, amountBDesired);
                liquidity = Math.min(amountA * _totalSupply / reserveA, amountB * _totalSupply / reserveB);
            }
        }
    }

    function quoteRemoveLiquidity(
        address tokenA,
        address tokenB,
        bool stable,
        uint liquidity
```



```
    ) external view returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        address _pair = IBaseV1Factory(factory).getPair(tokenA, tokenB, stable);

        if (_pair == address(0)) {
            return (0,0);
        }

        (uint reserveA, uint reserveB) = getReserves(tokenA, tokenB, stable);
        uint _totalSupply = erc20(_pair).totalSupply();

        amountA = liquidity * reserveA / _totalSupply; // using balances ensures pro-rata distribution
        amountB = liquidity * reserveB / _totalSupply; // using balances ensures pro-rata distribution

    }

    function _addLiquidity(
        address tokenA,
        address tokenB,
        bool stable,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal returns (uint amountA, uint amountB) {
        require(amountADesired >= amountAMin);
        require(amountBDesired >= amountBMin);
        // create the pair if it doesn't exist yet
        address _pair = IBaseV1Factory(factory).getPair(tokenA, tokenB, stable);
        if (_pair == address(0)) {
            _pair = IBaseV1Factory(factory).createPair(tokenA, tokenB, stable);
        }
        (uint reserveA, uint reserveB) = getReserves(tokenA, tokenB, stable);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = quoteLiquidity(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'BaseV1Router: INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = quoteLiquidity(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'BaseV1Router: INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function addLiquidity(
        address tokenA,
```

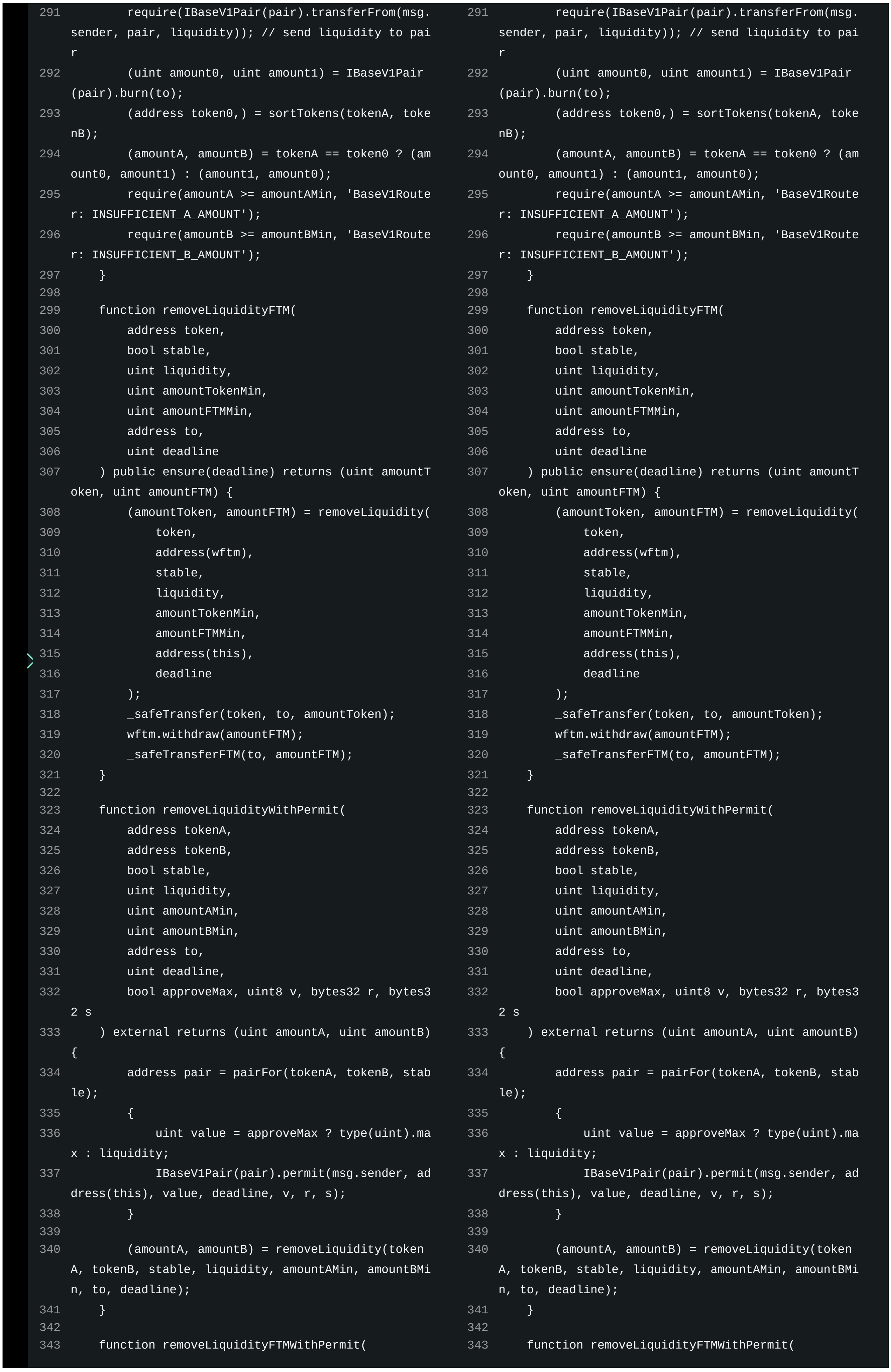
```
    ) external view returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        address _pair = IBaseV1Factory(factory).getPair(tokenA, tokenB, stable);

        if (_pair == address(0)) {
            return (0,0);
        }

        (uint reserveA, uint reserveB) = getReserves(tokenA, tokenB, stable);
        uint _totalSupply = erc20(_pair).totalSupply();

        amountA = liquidity * reserveA / _totalSupply; // using balances ensures pro-rata distribution
        amountB = liquidity * reserveB / _totalSupply; // using balances ensures pro-rata distribution

    }

    function _addLiquidity(
        address tokenA,
        address tokenB,
        bool stable,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal returns (uint amountA, uint amountB) {
        require(amountADesired >= amountAMin);
        require(amountBDesired >= amountBMin);
        // create the pair if it doesn't exist yet
        address _pair = IBaseV1Factory(factory).getPair(tokenA, tokenB, stable);
        if (_pair == address(0)) {
            _pair = IBaseV1Factory(factory).createPair(tokenA, tokenB, stable);
        }
        (uint reserveA, uint reserveB) = getReserves(tokenA, tokenB, stable);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = quoteLiquidity(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'BaseV1Router: INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = quoteLiquidity(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'BaseV1Router: INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function addLiquidity(
        address tokenA,
```

]



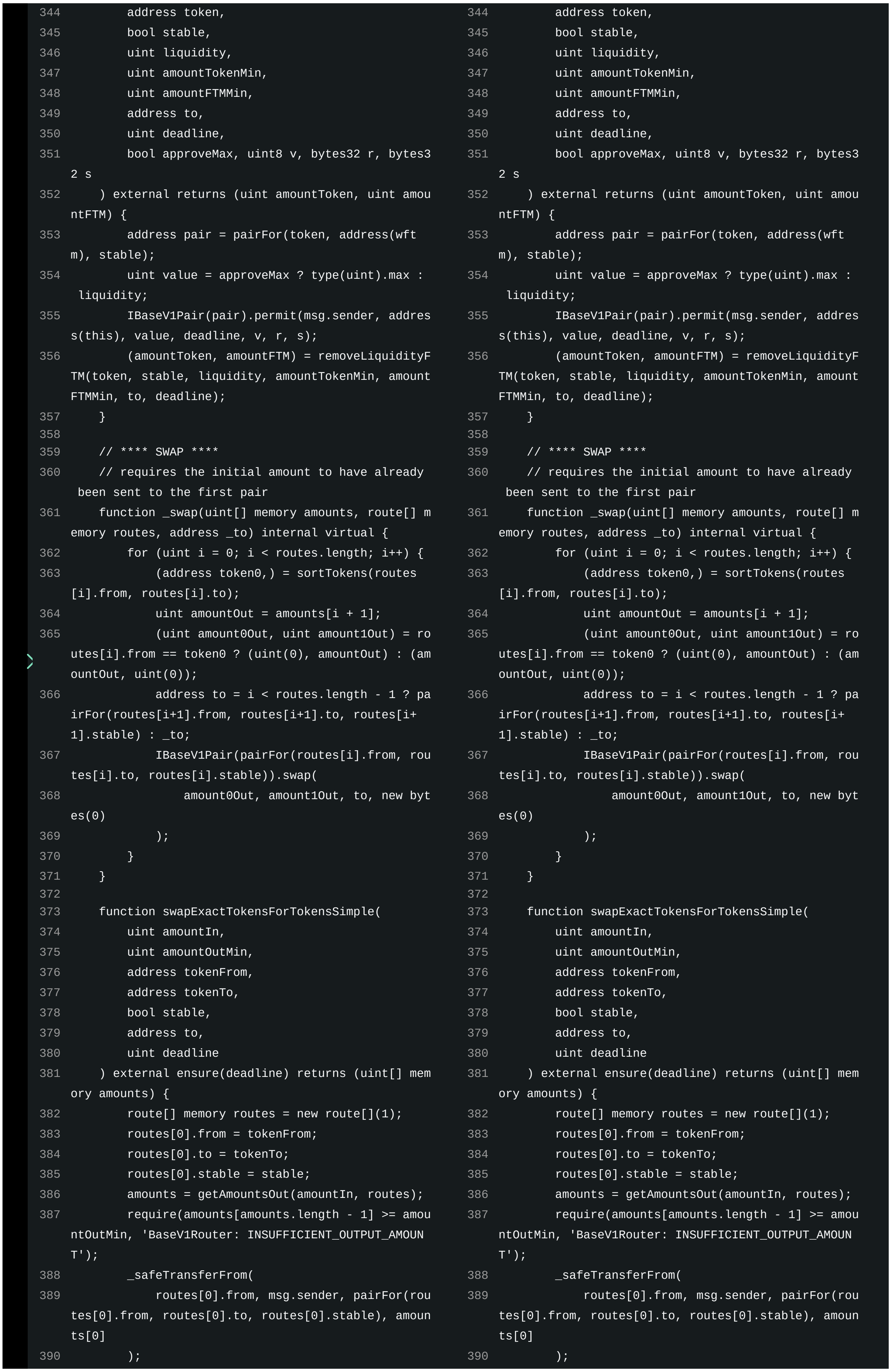
```
        address tokenB,
        bool stable,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, stable, amountADesired, amountBDesired, amountAMin, amountBMin);
        address pair = pairFor(tokenA, tokenB, stable);
        _safeTransferFrom(tokenA, msg.sender, pair, amountA);
        _safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IBaseV1Pair(pair).mint(to);
    }

    function addLiquidityFTM(
        address token,
        bool stable,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountFTMMin,
        address to,
        uint deadline
    ) external payable ensure(deadline) returns (uint amountToken, uint amountFTM, uint liquidity) {
        (amountToken, amountFTM) = _addLiquidity(
            token,
            address(wftm),
            stable,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountFTMMin
        );
        address pair = pairFor(token, address(wftm), stable);
        _safeTransferFrom(token, msg.sender, pair, amountToken);
        wftm.deposit{value: amountFTM}();
        assert(wftm.transfer(pair, amountFTM));
        liquidity = IBaseV1Pair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountFTM) _safeTransferFTM(msg.sender, msg.value - amountFTM);
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        bool stable,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public ensure(deadline) returns (uint amountA, uint amountB) {
        address pair = pairFor(tokenA, tokenB, stable);
```

```
        address tokenB,
        bool stable,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, stable, amountADesired, amountBDesired, amountAMin, amountBMin);
        address pair = pairFor(tokenA, tokenB, stable);
        _safeTransferFrom(tokenA, msg.sender, pair, amountA);
        _safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IBaseV1Pair(pair).mint(to);
    }

    function addLiquidityFTM(
        address token,
        bool stable,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountFTMMin,
        address to,
        uint deadline
    ) external payable ensure(deadline) returns (uint amountToken, uint amountFTM, uint liquidity) {
        (amountToken, amountFTM) = _addLiquidity(
            token,
            address(wftm),
            stable,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountFTMMin
        );
        address pair = pairFor(token, address(wftm), stable);
        _safeTransferFrom(token, msg.sender, pair, amountToken);
        wftm.deposit{value: amountFTM}();
        assert(wftm.transfer(pair, amountFTM));
        liquidity = IBaseV1Pair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountFTM) _safeTransferFTM(msg.sender, msg.value - amountFTM);
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        bool stable,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public ensure(deadline) returns (uint amountA, uint amountB) {
        address pair = pairFor(tokenA, tokenB, stable);
```



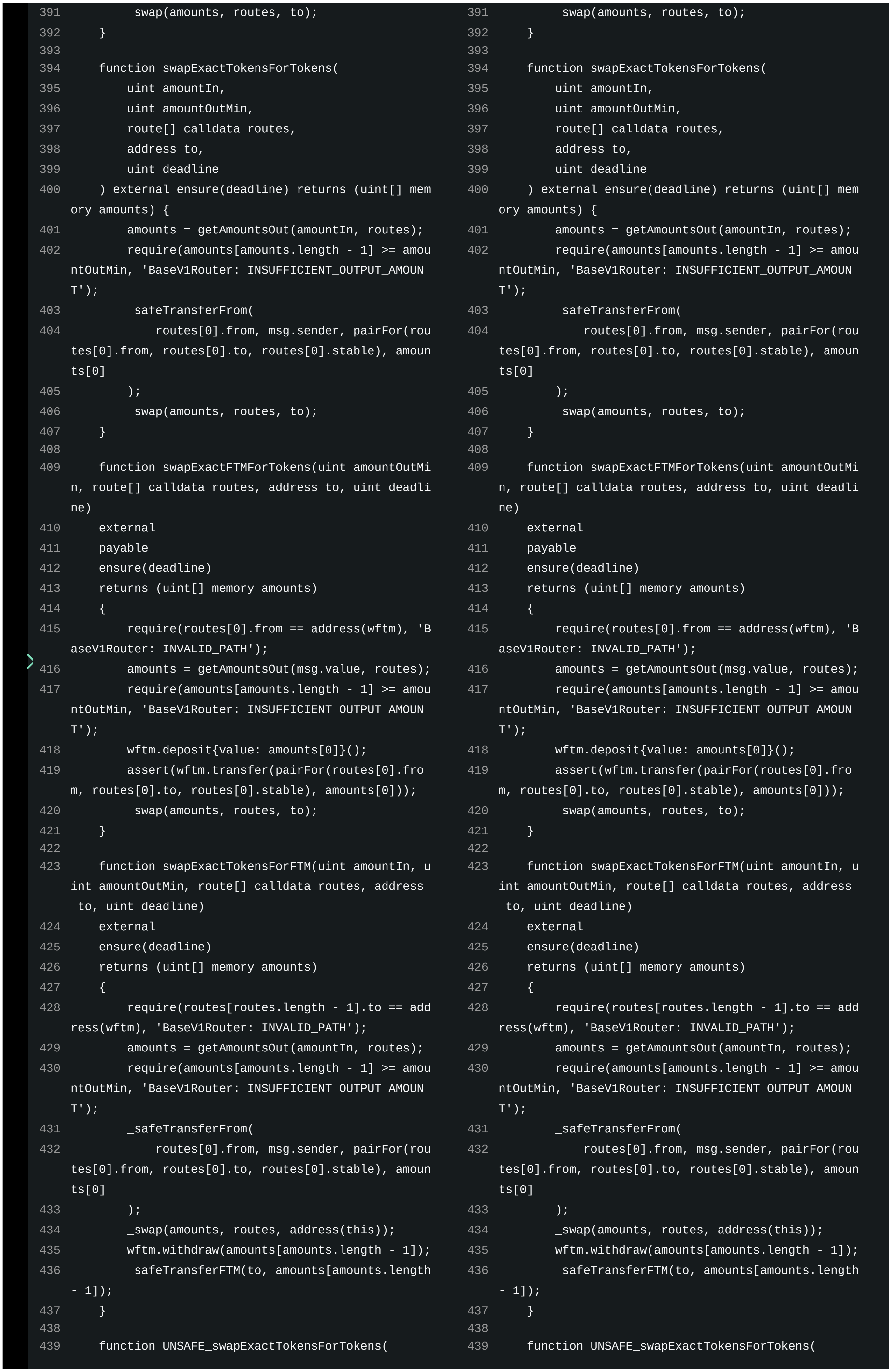
```
        require(IBaseV1Pair(pair).transferFrom(msg.sender, pair, liquidity)); // send liquidity to pair
        (uint amount0, uint amount1) = IBaseV1Pair(pair).burn(to);
        (address token0,) = sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'BaseV1Router: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'BaseV1Router: INSUFFICIENT_B_AMOUNT');
    }

    function removeLiquidityFTM(
        address token,
        bool stable,
        uint liquidity,
        uint amountTokenMin,
        uint amountFTMMin,
        address to,
        uint deadline
    ) public ensure(deadline) returns (uint amountToken, uint amountFTM) {
        (amountToken, amountFTM) = removeLiquidity(
            token,
            address(wftm),
            stable,
            liquidity,
            amountTokenMin,
            amountFTMMin,
            address(this),
            deadline
        );
        _safeTransfer(token, to, amountToken);
        wftm.withdraw(amountFTM);
        _safeTransferFTM(to, amountFTM);
    }

    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        bool stable,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB) {
        address pair = pairFor(tokenA, tokenB, stable);
        {
            uint value = approveMax ? type(uint).max : liquidity;
            IBaseV1Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        }

        (amountA, amountB) = removeLiquidity(tokenA, tokenB, stable, liquidity, amountAMin, amountBMin, to, deadline);
    }

    function removeLiquidityFTMWithPermit(
```

```
        require(IBaseV1Pair(pair).transferFrom(msg.sender, pair, liquidity)); // send liquidity to pair
        (uint amount0, uint amount1) = IBaseV1Pair(pair).burn(to);
        (address token0,) = sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'BaseV1Router: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'BaseV1Router: INSUFFICIENT_B_AMOUNT');
    }

    function removeLiquidityFTM(
        address token,
        bool stable,
        uint liquidity,
        uint amountTokenMin,
        uint amountFTMMin,
        address to,
        uint deadline
    ) public ensure(deadline) returns (uint amountToken, uint amountFTM) {
        (amountToken, amountFTM) = removeLiquidity(
            token,
            address(wftm),
            stable,
            liquidity,
            amountTokenMin,
            amountFTMMin,
            address(this),
            deadline
        );
        _safeTransfer(token, to, amountToken);
        wftm.withdraw(amountFTM);
        _safeTransferFTM(to, amountFTM);
    }

    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        bool stable,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB) {
        address pair = pairFor(tokenA, tokenB, stable);
        {
            uint value = approveMax ? type(uint).max : liquidity;
            IBaseV1Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        }

        (amountA, amountB) = removeLiquidity(tokenA, tokenB, stable, liquidity, amountAMin, amountBMin, to, deadline);
    }

    function removeLiquidityFTMWithPermit(
```



```
        address token,
        bool stable,
        uint liquidity,
        uint amountTokenMin,
        uint amountFTMMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountFTM) {
        address pair = pairFor(token, address(wftm), stable);
        uint value = approveMax ? type(uint).max : liquidity;
        IBaseV1Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountFTM) = removeLiquidityFTM(token, stable, liquidity, amountTokenMin, amountFTMMin, to, deadline);
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, route[] memory routes, address _to) internal virtual {
        for (uint i = 0; i < routes.length; i++) {
            (address token0,) = sortTokens(routes[i].from, routes[i].to);
            uint amountOut = amounts[i + 1];
            (uint amount0Out, uint amount1Out) = routes[i].from == token0 ? (uint(0), amountOut) : (amountOut, uint(0));
            address to = i < routes.length - 1 ? pairFor(routes[i+1].from, routes[i+1].to, routes[i+1].stable) : _to;
            IBaseV1Pair(pairFor(routes[i].from, routes[i].to, routes[i].stable)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }

    function swapExactTokensForTokensSimple(
        uint amountIn,
        uint amountOutMin,
        address tokenFrom,
        address tokenTo,
        bool stable,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint[] memory amounts) {
        route[] memory routes = new route[](1);
        routes[0].from = tokenFrom;
        routes[0].to = tokenTo;
        routes[0].stable = stable;
        amounts = getAmountsOut(amountIn, routes);
        require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
        _safeTransferFrom(
            routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
        );
```

```
        address token,
        bool stable,
        uint liquidity,
        uint amountTokenMin,
        uint amountFTMMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountFTM) {
        address pair = pairFor(token, address(wftm), stable);
        uint value = approveMax ? type(uint).max : liquidity;
        IBaseV1Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountFTM) = removeLiquidityFTM(token, stable, liquidity, amountTokenMin, amountFTMMin, to, deadline);
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, route[] memory routes, address _to) internal virtual {
        for (uint i = 0; i < routes.length; i++) {
            (address token0,) = sortTokens(routes[i].from, routes[i].to);
            uint amountOut = amounts[i + 1];
            (uint amount0Out, uint amount1Out) = routes[i].from == token0 ? (uint(0), amountOut) : (amountOut, uint(0));
            address to = i < routes.length - 1 ? pairFor(routes[i+1].from, routes[i+1].to, routes[i+1].stable) : _to;
            IBaseV1Pair(pairFor(routes[i].from, routes[i].to, routes[i].stable)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }

    function swapExactTokensForTokensSimple(
        uint amountIn,
        uint amountOutMin,
        address tokenFrom,
        address tokenTo,
        bool stable,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint[] memory amounts) {
        route[] memory routes = new route[](1);
        routes[0].from = tokenFrom;
        routes[0].to = tokenTo;
        routes[0].stable = stable;
        amounts = getAmountsOut(amountIn, routes);
        require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
        _safeTransferFrom(
            routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
        );
```



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

    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        route[] calldata routes,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint[] memory amounts) {
        amounts = getAmountsOut(amountIn, routes);
        require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
        _safeTransferFrom(
            routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
        );
        _swap(amounts, routes, to);
    }

    function swapExactFTMForTokens(uint amountOutMin, route[] calldata routes, address to, uint deadline)
    external
    payable
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(routes[0].from == address(wftm), 'BaseV1Router: INVALID_PATH');
        amounts = getAmountsOut(msg.value, routes);
        require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
        wftm.deposit{value: amounts[0]}();
        assert(wftm.transfer(pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]));
        _swap(amounts, routes, to);
    }

    function swapExactTokensForFTM(uint amountIn, uint amountOutMin, route[] calldata routes, address to, uint deadline)
    external
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(routes[routes.length - 1].to == address(wftm), 'BaseV1Router: INVALID_PATH');
        amounts = getAmountsOut(amountIn, routes);
        require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
        _safeTransferFrom(
            routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
        );
        _swap(amounts, routes, address(this));
        wftm.withdraw(amounts[amounts.length - 1]);
        _safeTransferFTM(to, amounts[amounts.length - 1]);
    }

    function UNSAFE_swapExactTokensForTokens(
```

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

    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        route[] calldata routes,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint[] memory amounts) {
        amounts = getAmountsOut(amountIn, routes);
        require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
        _safeTransferFrom(
            routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
        );
        _swap(amounts, routes, to);
    }

    function swapExactFTMForTokens(uint amountOutMin, route[] calldata routes, address to, uint deadline)
    external
    payable
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(routes[0].from == address(wftm), 'BaseV1Router: INVALID_PATH');
        amounts = getAmountsOut(msg.value, routes);
        require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
        wftm.deposit{value: amounts[0]}();
        assert(wftm.transfer(pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]));
        _swap(amounts, routes, to);
    }

    function swapExactTokensForFTM(uint amountIn, uint amountOutMin, route[] calldata routes, address to, uint deadline)
    external
    ensure(deadline)
    returns (uint[] memory amounts)
    {
        require(routes[routes.length - 1].to == address(wftm), 'BaseV1Router: INVALID_PATH');
        amounts = getAmountsOut(amountIn, routes);
        require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
        _safeTransferFrom(
            routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
        );
        _swap(amounts, routes, address(this));
        wftm.withdraw(amounts[amounts.length - 1]);
        _safeTransferFTM(to, amounts[amounts.length - 1]);
    }

    function UNSAFE_swapExactTokensForTokens(
        uint[] memory amounts,
        route[] calldata routes,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint[] memory) {
        _safeTransferFrom(routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]);
        _swap(amounts, routes, to);
        return amounts;
    }

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

    function _safeTransfer(address token, address to, uint256 value) internal {
        require(token.code.length > 0);
        (bool success, bytes memory data) =
        token.call(abi.encodeWithSelector(erc20.transfer.selector, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))));
    }

    function _safeTransferFrom(address token, address from, address to, uint256 value) internal {
        require(token.code.length > 0);
        (bool success, bytes memory data) =
        token.call(abi.encodeWithSelector(erc20.transferFrom.selector, from, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))));
    }
}

```

```
        uint[] memory amounts,
        route[] calldata routes,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint[] memory) {
        _safeTransferFrom(routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]);
        _swap(amounts, routes, to);
        return amounts;
    }

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

    function _safeTransfer(address token, address to, uint256 value) internal {
        require(token.code.length > 0);
        (bool success, bytes memory data) =
        token.call(abi.encodeWithSelector(erc20.transfer.selector, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))));
    }

    function _safeTransferFrom(address token, address from, address to, uint256 value) internal {
        require(token.code.length > 0);
        (bool success, bytes memory data) =
        token.call(abi.encodeWithSelector(erc20.transferFrom.selector, from, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))));
    }
}
```