1. Page 1

Contracts on Ethereum

In this assignment, you'll build a simple, Automated Market Maker (AMM) contract on Ethereum (similar to Uniswap).

The Ethereum Virtual Machine (EVM) executes byte code, and both Solidity and Vyper are high-level languages that can compile to EVM-compatible byte code. In this assignment, we'll use Solidity.

We won't actually deploy the contract on the Ethereum main net, or a test net, but instead we'll test the contract within the foundry testing environment.

Assignment

Conceptually, the lifecycle of an AMM has three parts

1. A "Liquidity Provider" deposits two types of tokens into the contract
2. Users trade one type of token for the other
3. The Liquidity Provider closes the contract and withdraws all tokens

The contract you write, using the skeleton provided in AMM.sol should have three external functions

1. provideLiquidity(tokenA_quantity, tokenB_quantity)
   Both tokenA_quantity and tokenB_quantity should be the quantities of each token that are being deposited.
   The sender corresponds to the address which provides liquidity (and therefore is the owner)
2. tradeTokens(sell_token, sell_quantity)
   sell_token should match either tokenA_addr or tokenB_addr, and sell_quantity should be the amount of that token being traded *to* the contract. The contract should calculate the amount of the other token to

1. Page 1

how it works here

3. withdrawLiquidity(recipient, amtA, amtB()
   If the message sender was the initial liquidity provider, this should send the amount of A or B tokens specified to the recipient address assuming the message sender has sufficent balance to cover the requested amounts, otherwise it should fail.

Exchange rate calculation

If the initial balances of the contract are A_i and B_i, then the "invariant" is $k = A_i \cdot B_i$.
Now, imagine a user deposits Δ_A tokens (of type A), if there were no fees, they would receive Δ_B tokens (of type B), where

$$k = (A_i + \Delta_A) \cdot (B_i - \Delta_B)$$

Your exchange should assess a trading fees. The fees are usually expressed in basis points and should be taken from the deposit side only. So if the user deposits Δ_A, you should calculate Δ_B as if the user had only deposited $\frac{10^4 - f}{10^4} \cdot \Delta_A$, where f is the fee in basis points.

Grading

Our autograder will use Foundry to test the functionality of your contract.

You can run the command

```
~/workspace/AMM$ forge test -vvv
```

This will run exactly the same tests that the autograder will run. If you're interested, you can see the tests in AMM.t.sol.

1. Page 1

EatTheBlocks has a nice tutorial on building a simple AMM that is similar to the one in this assignment.

ERC-20 Tokens

Although it is convenient to think of ERC-20 tokens as another on-chain currency like ETH, they behave somewhat differently.
Each type of ERC-20 token is handled by a smart contract, handles all token transfers and stores the user balances of that token.
This means that ERC-20 tokens cannot be “sent” in a transaction the same way you’d send ETH, but instead, ERC-20 tokens are “transferred” by calling the transfer function of the ERC-20 contract.
Your AMM will not need to “approve” tokens, but will need to use the transferFrom function, which means our autograder will have to approve the transfer before calling "tradeTokens".

Important Development Notes for ERC-20

OpenZeppelin provides templates that make it easy to create ERC20 compliant tokens. You will *not* have to create or deploy ERC20 tokens (our autograders will do that) but you must be familiar with their behavior so your AMM can transfer them.

One function of an ERC-20 contract is the “transferFrom” function which allows you to “pull” ERC-20 tokens from another user (instead of “transfer” which “pushes” tokens from the sender to the receiver). In order to prevent theft, the owner must first “approve” the receiver, before the “transferFrom” call.

Example

If you are having trouble, you can take a look at the "swap" function in the Uniswap v2 Pair contract. Uniswap is very well designed AMM, and their