Zadanie 2 – Solidity burza v decentralizovanom svete - dvojice

DISCLAIMER: Toto zadanie vychádza do značnej miery z open-source kurzu o blockchain technológiách fungujúcom na Univerzite v Stanforde.

Zadanie 2 je povinné! V rámci tohto projektu budete pracovať na webovom klientovi a dvoch Solidity kontraktoch na implementáciu decentralizovanej kryptomenovej burzy. Na konci bude mať vaša burza mnoho funkcií, ktoré majú plnohodnotné decentralizované burzy, ako napríklad Uniswap. Okrem toho vytvoríte vlastný ERC20 token, ktorý budete potom obchodovať na vašej burze.

Pri realizácii tohto projektu sa uistite, že píšete kód, ktorý je odolný voči útokom. Pamätajte, že útočníci môžu volať vaše smart kontrakty s ľubovoľnými vstupmi, nie len cez poskytnuté používateľské rozhranie. Preto používajte `require` príkazy na testovanie akýchkoľvek predpokladov zakomponovaných do vášho kódu.

Pre väčšinu študentov bude tento projekt pravdepodobne ich najkomplexnejším Solidity projektom. Preto vám odporúčame začať včas a klásť otázky. Prevedieme vás implementáciou decentralizovanej burzy v niekoľkých fázach a veríme, že váš konečný produkt vám dá niečo, na čo môžete byť hrdí! Poďme na to.

V rámci tohto vybudujeme aj používateľské rozhranie, ktoré počíta pre používateľa užitočné informácie a umožňuje neprogramátorom používať vašu DAppku.

1. Začíname

Prepokladáme, že máte nastavené prostredie `NodeJS, Hardhat, Solidity`. (ideálne využiť virtuálku, čo sme vám dali)

- Otvorte adresár štartovacieho kódu vo svojom obľúbenom IDE alebo textovom editore (niečo ako Sublime Text, Atom alebo Visual Studio Code fungujú dobre). Budete upravovať `contracts/token.sol` a `contracts/exchange.sol` na definovanie vašich kontraktov v Solidity a `web_app/exchange.js` na Javascript backend. Pohľad na ostatné súbory vám môže pomôcť pochopiť, ako Hardhat node a webový klient fungujú. V kódoch sú miesta označené funkciami na úpravu (alebo TODO) a jedine do troch vyššie uvedených súborov môžete pridať vaše pomocné funkcie. Prosím, neupravujte žiadne iné kódy ani neinštalujte ďalšie knižnice.
- Prečítajte si štartovací kód, dokumentáciu ethers.js, dokumentáciu Solidity a implementáciu kontraktu OpenZeppelin ERC-20. Dobre si premyslite celkový dizajn vášho systému pred začiatkom písania kódu. Aké údaje by sa mali uchovávať v blockchaine? Aký výpočet bude urobený v kontrakte vs. na klientovi?

- Po dokončení implementácie spustite `npx hardhat node` na vytvorenie localhost uzla. Ak je uzol spustený správne, v termináli by ste mali vidieť: *Started HTTP and WebSocket JSON-RPC server at https://localhost:8545*
- **Nasadenie kontraktov** (deployment): Otvorte si ďalšiu kartu alebo okno terminálu a dostaňte sa do adresára kódu. Spustite `npx hardhat run --network localhost scripts/deploy_token.js` na kompiláciu a nasadenie svojho kontraktu pre token. Po úspechu by ste mali na termináli vidieť túto správu: *Finished writing token contract address: ...* Skopírujte túto hodnotu a vložte ju do poľa adresy `tokenAddr` v `contracts/exchange.sol`.
- Spustite `npx hardhat run --network localhost scripts/deploy_exchange.js` na nasadenie kontraktu burzy. Po úspechu by sa mala zobraziť správa: *Finished writing exchange contract address: ...* Uložte si túto adresu pre ďalší krok.
- Aktualizujte adresu kontraktu a rozhranie ABI v súbore `web_app/exchange.js`. Aktualizujte premenné `const token_address` a `const exchange_address` pomocou dvoch adries kontraktov z predchádzajúcich bodov. Na rozdiel od Solidity, adresy v Javascripte tak prísne kontrolované. ABI je možné skopírovať z `artifacts/contracts/token.sol/token.json` a `artifacts/contracts/exchange.sol/exchange.json`. Ak chcete správne aktualizovať ABI, prosím skopírujte celý zoznam za poľom „abi“, počnúc hranatou zátvorkou. Adresu ste uložili v predchádzajúcom kroku. Uistite sa, že adresa kontraktu je ako string.
- Otvorte súbor `web_app/index.html` vo svojom prehliadači. Kým nedokončíte implementáciu v sekcii 3 (`contracts/token.sol`), v konzole prehliadača budete mať chybu. Inak sa ale môžete pohrať so stránkou a spustiť `sanityCheck`! Ďalšie podrobnosti nájdete v sekcii 10.

2. Komponenty

Projekt má tri hlavné časti:

- `contracts/token.sol` je kontrakt napísaný v Solidity, ktorý bude nasadený na blockchain. Ak chcete vytvoriť a nasadiť svoj vlastný token, budete ho musieť upraviť. Podrobnosti o implementácii nájdete v sekcii 3.
- `contracts/exchange.sol` je ďalší kontrakt napísaný v Solidity a nasadený na blockchain. Budete ho musieť upraviť, aby ste vytvorili decentralizovanú burzu (DEX),

nazývanú aj automatizovaný tvorca trhu (AMM), po vzore Uniswap V1. Podrobnosti o implementácii nájdete v sekcii 4.

- `web_app/exchange.js` je klient spustený lokálne vo webovom prehliadači napísaný v JavaScripte. Pozoruje blockchain pomocou knižnice `ethers.js` a môže volať funkcie v kontrakoch `token.sol` a `exchange.sol`. Ďalšie podrobnosti nájdete v sekcii 4.

Súbor `web_app/index.html` po otvorení vo vašom prehliadači (najlepšie funguje v Chrome) vám umožňuje prístup k používateľskému rozhraniu, prostredníctvom ktorého môžete otestovať svoju burzu. Na tejto stránke si môžete vybrať adresu a pridať likviditu, odstrániť likviditu a vymeniť svoj token za ETH a naopak. Tento a iné súbory by sa nemali upravovať. Mali by ste upraviť iba `token.sol, exchange.sol a exchange.js.`

3. Vytvorenie a nasadenie vlastného tokenu

V prvej časti tohto projektu si vytvoríte a nasadíte svoj vlastný ERC-20 token. ERC-20 je štandardom na implementáciu zameniteľných tokenov. Našťastie pre nás, veľká časť kódu pre štandard ERC-20 už bola napísaná a je open source. V tomto projekte použijeme štandardnú implementáciu ERC-20 z projektu OpenZeppelin. Uistite sa, že rozumiete štandardu ERC-20, ako aj implementáciám OpenZeppelin ERC-20 a Ownable. Keď si prečítate štartovací kód, vykonajte nasledujúce kroky v `contracts/token.sol` a `web_app/exchange.js`:

- Vymyslite zábavné (ale vhodné!) meno pre svoj token. Akékoľvek meno je fajn, tak sa pokojne zabavte a vymyslite niečo kreatívne. Nastavte súkromný string `_name` v token.sol ako názov vášho tokenu. Okrem toho aktualizujte premennú `token_name` v hornej časti súboru `exchange.js` tak, aby mala rovnaký názov.
- Vymyslite krátky symbol pre svoj token (napr. ETH namiesto Ethereum). Nastavte súkromný string `_symbol` na túto skratku a aktualizujte premennú `token_symbol` v hornej časti `exchange.js`.
- Implementujte funkcie `mint(uint amount)` a `disable_mint(). mint` je verejná funkcia, ktorá vytvára isté množstvo (amount) tokenov. Aj keď kontrakt OpenZeppelin ERC-20 neposkytuje funkciu mint(), môžete na to vhodne zavolať `_mint(). disable_mint` spôsobí, že volanie mint() už nikdy nebude úspešné. Ako taká, vaša funkcia `mint` musí zlyhať, ak bolo zavolané `disable_mint`. Všimnite si tiež modifikátor `Ownable` na oboch funkciách, vďaka ktorému môže jednu z nich zavolať iba vlastník kontraktu. Paradigma `mint()/disable_mint()` je jednoduchý spôsob počiatočného generovania tokenov a zaručenia, že maximálne množstvo tokenov ostane konštantné po vytvorení všetkých tokenov.

- Nasaďte svoj token kontrakt. Skopírujte adresu a ABI kontraktu do premenných `token_address` a `token_abi` v `exchange.js`.
- Nakoniec skopírujte adresu nasadeného token kontraktu do premennej `tokenAddr` v `exchange.sol`. **Zakaždým, keď robíte nasadenie svojho token kontraktu musíte tento krok zopakovať.**

Po dokončení vyššie uvedených krokov by ste mali mať svoj token nasadený na Hardhat! Teraz môžete skúsiť nasadiť aj svoju burzu. Skopírujte adresu a ABI token kontraktu do premenných `exchange_address` a `exchange_abi` v súbore `exchange.js`. Všetky funkcie okrem počiatočného nastavenia poolu (ktoré sme pre vás implementovali) budú chýbať. To znamená, že ak ste správne dokončili sekciu 2 a proces nasadenia token kontraktu, nemali by ste vidieť žiadne chyby v konzole vášho prehliadača a mali by ste vidieť výmenný kurz 1:1 medzi ETH a vaším tokenom. Na obrazovke „Current Liquidity“ by malo byť 5000 ETH a 5000 vašich tokenov.

4. Nastavenie vlastnej jednoduchej burzy

V tejto časti zadania implementujete základnú funkcionalitu vašej kryptomenovej burzy. Burza je modelovaná podľa Uniswap V1. Vaša burza umožní iba výmeny (swapy) medzi vaším tokenom a testovacím ETH. Zmeny v tejto časti ovplyvnia predovšetkým dva súbory: `exchange.js` a `exchange.sol`. Oboznámte sa so štartovacím kódom týchto súborov.

Decentralizovaná burza pozostáva z dvoch typov účastníkov: poskytovateľov likvidity (LP) a obchodníkov. Pre daný výmenný pool medzi dvoma tokenmi poskytujú poskytovatelia likvidity určitú rovnakú hodnotu oboch typov likvidity (vo vašom prípade ETH a váš vlastný token). Keď obchodníci swapujú medzi dvoma menami, pridajú do poolu likvidity určitú sumu jednej meny a z fondu sa im pošle rovnaká hodnota druhej meny. Výmenný kurz medzi týmito dvoma menami je určený vzorcom:

Nech x je množstvo meny A, ktorá je v liquidity poole, a nech y je množstvo meny B. Nech k je nejaká konštanta. Po každej výmene musí platiť, že

$$x * y = k$$

Počas každého swapu musí burza odoslať správnu sumu swapovanej meny tak, aby pool vždy zostal na konštantnej krivke. Cena meny B v mene A sa môže vypočítať ako x/y, zatiaľ čo cena meny A v mene B sa môže vypočítať ako y/x. Každý swap tak zmení výmenný kurz. To dáva zmysel, keďže každý swap je ukazovateľom dopytu po danej mene. Účinky každého swapu na cenu mien budú relevantné v sekcii 6.

Keď poskytovateľ likvidity pridá likviditu, musí poskytnúť rovnakú hodnoty meny A aj B, ako je určené aktuálnym výmenným kurzom. Upozorňujeme, že pridaním likvidity sa nezmení

výmenný kurz medzi menami, ale zvýši sa hodnota konštanty k. Podobne, keď si poskytovateľ likvidity ide vybrať svoju likviditu, musí stiahnuť rovnakú hodnotu meny A aj meny B. Preto sa k zníži, ale výmenný kurz zostane rovnaký.

To má ďalší pozoruhodný dôsledok: keďže poskytovateľ likvidity môže vybrať len rovnaké hodnoty každej meny, v skutočnosti nie je oprávnený vybrať svoju presnú počiatočnú investíciu (v zmysle množstva každého tokenu). Poskytovanie likvidity je skôr analogické s vlastnením percentuálneho podielu z poolu likvidity, ktorý je potom poskytovateľ oprávnený neskôr vybrať. Poskytovateľ likvidity, ktorý poskytol 10 % meny A a meny B, je oprávnený vybrať 10 % z každej z rezerv pre tieto meny (za predpokladu, že jeho percento nebolo znížené inými poskytovateľmi pridaním likvidity), aj keď 10 % presne nezodpovedá množstvám každého tokenu, ktorý poskytli. Týmto spôsobom môžu poskytovatelia likvidity utrpieť stratu (impermanent loss), ak hodnota toho, čo majú právo vybrať, je nižšia ako hodnota, ktorú pôvodne investovali.

S ohľadom na vyššie uvedené teraz implementujete základnú funkcionalitu vašej burzy. Postaráme sa o inicializáciu poolu za vás implementáciou a volaním funkcie `createPool`. Na inicializáciu fondu zoberieme ETH a tokeny z prvej adresy a pri *sledovaní poskytovateľov likvidity nemusíte túto počiatočnú sumu sledovať*. **Aby mohli iné adresy získať tokeny a/alebo poskytnúť likviditu, musia najprv vymeniť tokeny na burze**. V `exchange.sol` implementujte nasledujúce funkcie:

- `function addLiquidity() external payable`: Pridajte likviditu do poolu, ak má poskytovateľ dostatok ETH a tokenov (inak by transakcia zlyhala). Volajúci pošle ETH do kontraktu, ku ktorému sa dá dostať pomocou `msg.value`. Táto funkcia by tiež mala preniesť ekvivalentné množstvo tokenov na základe aktuálneho výmenného kurzu z adresy odosielateľa do kontraktu (pomocou metódy `transfer` alebo `transferFrom`) a podľa toho aktualizovať stav burzy. Transakcia musí zlyhať, ak sú finančné prostriedky poskytovateľa nedostatočné. V sekcii 9 nájdete rady, ako najlepšie sledovať likviditu.
- `function removeLiquidity(uint amountETH) public payable`: Odstráňte z poolu určené množstvo likvidity (ak je poskytovateľ oprávnený odobrať dané množstvo likvidity) a podľa toho aktualizujte stav burzy. `amountETH` je číselná hodnota ETH, ktorú chce poskytovateľ likvidity vybrať, takže po prijatí tokenov a ETH by mal dostať celkovú hodnotu ekvivalentnú 2 * `amountETH`. Nezabudnite zodpovedajúcim spôsobom aktualizovať množstvo likvidity poskytovanej jednotlivými poskytovateľmi likvidity. Táto funkcia by mala zlyhať, ak sa používatelia pokúsia

odstrániť viac likvidity, ako majú nárok, alebo ak sa pokúsia vyčerpať rezervy ETH alebo tokenov na 0.

- `function removeAllLiquidity() external payable`: Odstráňte maximálne množstvo likvidity, ktoré môže odosielateľ odobrať, a podľa toho aktualizujte stav burzy. Okrem toho nezabudnite aktualizovať množstvo likvidity poskytovanej každým poskytovateľom likvidity. Aj táto funkcia by mala zlyhať, ak poskytovateľ likvidity vyčerpá rezervy ETH alebo tokenov na 0.
- `function swapTokensForETH(uint amountTokens) external payable`: Vymeňte dané množstvo tokenov za ekvivalentnú hodnotu ETH a podľa toho aktualizujte stav burzy. Ak poskytovateľ nemá dostatok tokenov na swap, transakcia by mala zlyhať. Okrem toho, ak by dokončenie swapu úplne odstránilo všetky ETH z poolu, transakcia by mala zlyhať kvôli zabráneniu nulového stavu ETH a (preto) nedefinovaného výmenného kurzu. Uistite sa, že vždy necháte v poole aspoň 1 ETH a 1 token.
- `function swapETHForTokens() external payable`: Vymeňte dané množstvo ETH za ekvivalentnú hodnotu vo vašom tokene a podľa toho aktualizujte stav burzy. Podobne ako pri addLiquidity(), odosielateľ pošle ETH do kontraktu, ku ktorému sa dá dostať cez `msg.value`. Ak by dokončenie swapu úplne odstránilo všetky tokeny z poolu, transakcia by mala zlyhať aby burza nemala nula tokenov a (preto) nedefinovaný výmenný kurz. Uistite sa, že vždy necháte v poole aspoň 1 ETH a 1 token.

V každej z vyššie uvedených funkcií sa uistite, že upravujete `token_reserves`, `eth_reserves` a/alebo k správnym spôsobom tak, aby výmena bola vždy na konštantnej krivke opísanej vyššie. Okrem toho sa uistite, že funkcie zlyhajú, keď volajúci nemá dostatok finančných prostriedkov. Nakoniec nezabudnite nastaviť adresu `tokenAddr` ako adresu vášho nasadeného kontraktu. Teraz môžete spustiť `npx hardhat run --network localhost scripts/deploy_exchange.js` na odstránenie chýb a nasadenie vášho kódu.

Riešenie chýb zaokrúhľovania: Výmena tokenov podľa pravidla konštantnej krivky zahŕňa vykonávanie operácií, ako sú sčítanie a delenie, čo nevyhnutne vedie k chybám zaokrúhľovania. Je dôležité zabezpečiť, aby sa tieto chyby nemohli objaviť. To by mohlo viesť k stratám pre poskytovateľov likvidity, ak by sa napríklad pri každom swape z poolu vybralo o niečo viac tokenov, ako sa očakávalo.

V dôsledku toho sa súčin x*y nemusí presne rovnať k kvôli niektorým chybám zaokrúhľovania - čo nie je problém - ale keďže k zostáva striktne nemodifikované, tieto chyby sa neznásobujú, ale skôr sa samy kompenzujú, keď je veľa swapov. Chyby zaokrúhľovania sa

môžu vyskytnúť aj pri pridávaní alebo odoberaní likvidity, pretože váš token a ETH nie je možné rozdeliť na viac ako 1 jednotku (prinajmenšom v tomto projekte, ďalšie podrobnosti v sekcii 8). *Neočakávame, že budete v tomto projekte riešiť všetky typy zaokrúhľovacích chýb*. Je teda v poriadku, ak sa x*y odchyľuje od k pri vykonávaní aritmetiky, ako aj iných podobných zaokrúhlení pri výpočte likvidity. Napriek tomu je stále dôležité aktualizovať k na vhodných miestach.

5. Implementácia backendu

Po dokončení implementácie funkcií kontraktov implementujte do `exchange.js` nasledujúce funkcie. Premennú `maxSlippagePct` môžete zatiaľ ignorovať – použije sa v sekcii 6. Vo väčšine prípadov by to len zavolalo funkcie tokenu a burzy, ktoré ste napísali vyššie.

- `async function addLiquidity(amountEth, maxSlippagePct),`
- `async function removeLiquidity(amountEth, maxSlippagePct),`
- `async function removeAllLiquidity(maxSlippagePct),`
- `async function swapTokensForETH(amountToken, maxSlippagePct),`
- `async function swapETHForTokens(amountEth, maxSlippagePct).`

Kód kontraktu môžete zavolať pomocou `await contract.functionName(args)` alebo `await contract.connect(anotherSigner).functionName(args).` Ak potrebujete ďalšiu pomoc so syntaxou, dôrazne vám odporúčame pozrieť si dokumentáciu ethers.js, ako aj sekciu 9, kde nájdete niekoľko tipov.

Po úplnej implementácii kontraktu a zodpovedajúceho JavaScript kódu aktualizujte premenné `token_abi` a `exchange_abi` v hornej časti súboru a skopírujte adresy kontraktov do premenných `token_address` a `exchange_address` v súbore `exchange.js`. Pri kopírovaní ABI nezabudnite zahrnúť krajné zátvorky. Ak obnovíte stránku index.html, teraz by ste mali byť schopní poskytnúť likviditu, odstrániť likviditu a vykonávať swapy.

6. Manipulácia s oddchýlkou

S našou burzou je významný problém, keďže sme ju implementovali v sekcii 3 a neberie do úvahy odchýlku alebo tzv. „sklz“ (slippage). Pripomeňme, že pri každom swape na decentralizovanej burze sa cena každého aktíva mierne posunie. Keďže sa mnohí používatelia môžu pokúšať zameniť menu naraz na decentralizovanej burze, môže dôjsť k posunu vo výmennom kurze medzi odoslaním swapovej transakcie a skutočným spracovaním tejto transakcie. Táto odchýlka vo výmennom kurze medzi kótovanou cenou burzy a skutočnou cenou sa nazýva „sklz“. Pri obchodovaní s nestálymi aktívami je to obzvlášť znepokojujúce. Napríklad, ak používateľ odošle swapovú transakciu na výmenu určitej sumy meny A za menu

B a potom cena meny B dramaticky vzrastie z kótovanej ceny, používateľ si možno nebude želať dokončiť swapovú transakciu.

Okrem toho nedostatočná manipulácia predstavuje pre používateľov hrozbu na základe typu útoku známeho ako sendvičový útok. Sendvičový útok funguje nasledovne:

- Alice predloží swapovú transakciu na výmenu veľkého množstva meny A na menu B.
- Útočník vidí Alicinu transakciu a predbehne ju s veľkým nákupom meny B, čím sa zvýši cena aktíva B.
- Alica kúpi menu B za novú vyššiu cenu, čím ešte viac zvýši cenu meny B.
- Útočník potom okamžite predá všetku svoju novozískanú menu B za vyššiu cenu, čím dosiahne rýchly zisk.

Zraniteľnosť voči sendvičovým útokom je pre používateľov decentralizovanej burzy zlá, pretože neustále platia vyššie výmenné kurzy, než je skutočná hodnota aktív. Preto je dôležité, aby sme našu burzu inovovali, aby správne zvládala odchýlku a bránila sa proti sendvičovým útokom.

Najbežnejšou obranou proti sendvičovým útokom je umožniť používateľom nastaviť určitú maximálnu odchýlku pri odosielaní transakcie. Tento parameter, zvyčajne percento, spôsobí zlyhanie transakcie, ak sa cena aktív zmenila o viac, než je maximálny povolený stav. To obmedzuje škody, ktoré môžu spôsobiť sendvičové útoky, a chráni používateľov, ktorí vymieňajú volatilné aktíva. Ak chcete implementovať požiadavku na maximálnu odchýlku, vykonajte tieto kroky:

- V `exchange.sol` aktualizujte svoje funkcie `swapTokensForETH` a `swapETHForTokens` tak, aby vzali do úvahy parameter `uint min_exchange_rate`. V prípade potreby môžete vziať do úvahy aj iné parametre. Počas výmeny by swap mal zlyhať, ak sa aktuálna cena nového aktíva (tj. aktíva, na ktoré používateľ vymieňa) zvýšila na viac, než je maximálny výmenný kurz. Upozorňujeme, že zníženie ceny aktíva je pre používateľa dobré, takže v takom prípade nemusíme zlyhať.
- Aktualizujte svoje funkcie `addLiquidity`, `removeLiquidity` a `removeAllLiquidity` tak, aby vzali do úvahy parametre `uint min_exchange_rate` a `uint max_exchange_rate`. V prípade potreby môžete vziať do úvahy aj iné parametre. Počas poskytovania likvidity, transkacia zlyhá, ak sa aktuálna cena nového aktíva (tj. aktíva, na ktoré používateľ vymieňa) zvýšila na viac, než je maximálny výmenný kurz alebo znížila na menej než je minimálny. Náhle zmeny cien oboma smermi môžu poskytovateľov vystaviť strate predtým, ako vložia svoju

likviditu, a preto chceme, aby poskytovatelia likvidity špecifikovali maximálny a minimálny výmenný kurz.

- Teraz aktualizujte svoj súbor `exchange.js`, aby ste mohli komunikovať s kontraktom o max/min výmenných kurzoch. K dispozícii je parameter `maxSlippagePct`, ktorý predstavuje maximálnu povolenú percentuálnu zmenu ceny pred zlyhaním transakcie. Pri testovaní a v rozhraní prehliadača sa tento parameter odovzdáva ako int, nie ako float – tj. 4 % sa posiela ako 4, nie 0,04. Tento parameter možno použiť v každej JavaScript funkcii na výpočet správnych hodnôt pre maximálny výmenný kurz a/alebo minimálny výmenný kurz, ktoré sa potom môžu preniesť do kontraktu. Tu môže byť užitočná funkcia `getPoolState`, ktorú máte k dispozícii.

Ako vždy, po aktualizácii kontraktu nezabudnite znova skompilovať, nasadiť a skopírovať nové rozhranie ABI a adresu do premenných v hornej časti súboru `exchange.js`. V tomto bode môžete tiež odkomentovať funkciu sanityCheck() a skontrolovať svoju implementáciu. Ďalšie podrobnosti o kontrole sanityCheck nájdete v sekcii 10.

7. Odmeňovanie poskytovateľov likvidity – BONUS za max 3 body

Po dokončení vyššie uvedených sekcií máte teraz fungujúcu burzu, ktorá umožňuje používateľom obmedziť množstvo odchýlky, ktoré chcú tolerovať! Je tu však ešte jeden veľký problém. Niekoľkokrát sme diskutovali o tom, ako poskytovatelia likvidity podstupujú riziko v podobe straty (impermanent loss). To znamená, že hodnota ich likviditného podielu sa môže znížiť, ak sa zmení cena niektorého z aktív. V praxi, keďže mnohé kryptomeny sú dosť volatilné, ide o úroveň rizika, ktorú by žiadny poskytovateľ likvidity nebol ochotný podstúpiť zadarmo.

Preto musíme motivovať poskytovateľov likvidity, aby dali likviditu do poolu. V reálnych burzách sú poskytovatelia likvidity motivovaní poskytovať likviditu, pretože dostávajú malý poplatok z každej swapovej transakcie. Tieto poplatky sa automaticky reinvestujú do poolu likvidity v mene každého poskytovateľa likvidity. Keď si poskytovateľ ide vybrať svoju likviditu, suma, ktorú má právo vybrať, zahŕňa všetky poplatky, ktoré mu boli priznané od poskytnutia likvidity.

Teraz zavediete rovnakú schému odmeňovania pre poskytovateľov likvidity. **Môžete si slobodne navrhnúť svoj vlastný mechanizmus, pokiaľ spĺňa požiadavky uvedené na konci tejto sekcie.** Dôrazne však odporúčame nasledujúci vysvetlený postup:

- Každý poskytovateľ likvidity vlastní zlomok f poolu. Tento zlomok je uložený v smart kontrakte pre každého poskytovateľa likvidity (tj. pole uchováva pre každú adresu číslo ako časť poolu vlastneného touto adresou). Pre túto úlohu nastavíme swapový poplatok

na 3 %, reprezentované premennými `swap_fee_numerator` a `swap_fee_denominator.` Vlastnícke podiely poskytovateľov likvidity by mali byť pri vykonávaní swapov nezmenené, pretože odmeny za likviditu sa rozdeľujú na základe percenta vlastníctva každého poskytovateľa. Všimnite si, že kvôli poplatkom celková likvidita $l = \sqrt{xy}$ rastie, a tým by sa k trochu posunulo. To vedie k ziskom pre poskytovateľov likvidity, ak sa výmenný kurz nezmení, a slúži na boj proti impermanent loss.

- Ako bolo uvedené vyššie, vďaka odmenám za likviditu sa k teraz mierne zmení pri každom swape. Pre účely tohto projektu by ste však mali **meniť k len pri pridávaní alebo uberaní likvidity**. Nemusíte sa teda obávať týchto malých posunov k v dôsledku zmeny x*y počas swapovania.
- Keď poskytovateľ likvidity vyberie svoju likviditu, získa zlomok f oboch tokenov v poole zodpovedajúci ich podielu vlastníctva, ako aj ich odmeny. Ostatní poskytovatelia likvidity majú zodpovedajúcim spôsobom zvýšený podiel vlastníctva. Súčet všetkých zlomkov by mal byť 1.
- Keď poskytovateľ likvidity pridá likviditu do poolu, získa vlastnícky podiel (zlomok) f na poole rovnajúci sa podielu jeho tokenov v novom stave poolu. Ostatní poskytovatelia likvidity majú zodpovedajúcim spôsobom zmenšený podiel vlastníctva. Opäť platí, že súčet všetkých zlomkov by mal byť 1.

Ak ste implementovali sledovanie likvidity pomocou zlomkov, potom by táto časť nemala vyžadovať príliš veľa práce navyše. Prípadne akceptujeme akýkoľvek dizajn, ktorý spĺňa požiadavky uvedené nižšie.

Požiadavky na likviditu odmien:

- Váš pool musí účtovať osobe vykonávajúcej swap nejaký nenulový percentuálny poplatok za každú swapovú transakciu[1]. Túto hodnotu môžete definovať pomocou `swap_fee_enumerator` a `swap_fee_denominator`, ako bolo vysvetlené vyššie.
- Keď dôjde k swapu, hodnota tokenov alebo ETH odoslaných obchodníkovi by sa mala rovnať (1 - p) násobku hodnoty aktív, ktoré swapujú, kde p je percentuálny poplatok pre poskytovateľov likvidity. Napríklad, ak je poplatok 1% a používateľ zamieňa 100 ETH za vaše tokeny, mal by sa mu poslať iba ekvivalent 99 ETH.
- Keď sa počas swapu vyberie poplatok, mal by byť rozdelený medzi poskytovateľov likvidity tak, aby každý poskytovateľ mohol neskôr vybrať svoj pôvodný podiel plus

[1] Pre informáciu, predvolený poplatok na Uniswap je 0,3 %, zatiaľ čo centralizované burzy si zvyčajne účtujú za swap meny približne 1-4 %.

poplatky. Poplatky by sa mali rozdeliť proporcionálne na základe zlomkového podielu poskytovateľov na poole likvidity v čase, keď sa swap uskutočnil. Bolo by napríklad nesprávne, ak by poskytovateľ likvidity, ktorý poskytol polovicu všetkej likvidity v poole v čase t, mohol vybrať polovicu všetkých poplatkov vybratých pred časom t. Poskytovatelia likvidity by okrem volania removeLiquidity nemali robiť žiadne ďalšie kroky na uplatnenie svojich poplatkov. Okrem toho by sa odmeny za likviditu nemali posielať z burzy poskytovateľom zakaždým, keď sa uskutoční swap, pretože by to bolo v praxi neúmerne drahé.

- Pri rozhodovaní sa medzi rôznymi možnosťami dizajnu vám odporúčame zvoliť si riešenie, ktoré minimalizuje náklady na poplatky. Nebudeme hodnotiť striktne podľa spotreby poplatkov; budete však musieť zdôvodniť svoje rozhodnutia o dizajne v dokumente v sekcii 6.

Po navrhnutí a implementácii vyššie uvedenej časti by ste mali mať plne funkčnú burzu! Gratulujeme! Otestujte svoje funkcie pomocou poskytnutého používateľského rozhrania v súbore index.html alebo napíšte testovací kód v jazyku JavaScript. Realizácia tohto projektu predstavuje veľmi pôsobivý úspech, takže sa potľapkajte po pleci. V skutočnosti, s určitými bezpečnostnými úpravami, môžete nasadiť svoj token aj svoju burzu do siete Ethereum, a tak mať burzu, ktorú môžete volať svojou vlastnou!

8. Poznámky k Solidity a Javascript číslam

Na rozdiel od väčšiny programovacích jazykov, Solidity nepodporuje aritmetiku s pohyblivou rádovou čiarkou. Všetky tokeny ERC-20 teda sledujú premennú *decimals*, ktorá udáva, na koľko desatinných miest vľavo sa majú čísla interpretovať. Napríklad ether používa 18 desatinných miest, takže 1 ETH by bolo v kontrakte uvedené ako 10^{18} . Podobne 1 wei = 10^{-18} ETH, takže 1 wei je reprezentovaný len ako 1. Bohužiaľ, Javascript má tiež limit na to, aké veľké môžu byť celé čísla: Numbers.MAX_INT = 9 * 10^{15}. Aby sme vyvážili tieto dva rozdiely, na našej burze inicializujeme pool tak, aby mal 10^{10} tokenov, čo znamená, že pool začína s 10^{-8} ETH a 10^{-8} vašimi tokenmi.

Dôsledkom toho, že Solidity nepodporuje operácie float je, že všetky desatinné čísla budú skrátené. Napríklad 5 / 2 = 2. Preto je potrebná premenná *decimals*, pretože reprezentácia 5 ETH ako 5 * 10^{18} vedie k 5 * 10^{18} / 2 = 25 * 10^{16}, a keďže vieme, že existuje 18 desatinných miest, môžete vidieť, že to zodpovedá 2,5 ETH podľa želania. V niektorých prípadoch však môže dôjsť k podtečeniu, najmä ak je čitateľ menší ako menovateľ vo výpočtoch a teda vedie k nule. V takom prípade by ste si mali dávať pozor na poradie operácií, ako je násobenie alebo

sčítanie najprv pred rozdelením. Vo všeobecnosti je dobré odložiť delenie na čo najneskôr, aby ste predišli tejto chybe zaokrúhľovania.

V tomto projekte nebudeme testovať pretečenie (overflow) a pokúsili sme sa zo všetkých síl odstrániť všetky desatinné miesta. Preto pri testovaní vašej burzy vyberieme hodnoty, pri ktorých nepresiahneme INT_MAX Javascriptu. Vašu implementáciu však skontrolujeme, aby sme sa uistili, že ste pri burze rátali s underflow.

9. Rady ku implementácii

Aj keď je celkový návrh vašich kontraktov je otvorený, tu je niekoľko rád, ktoré vám môžu pomôcť zefektívniť proces implementácie:

- **Sledujte skôr podiely poskytovateľov likvidity ako absolútne hodnoty**. Napríklad, ak je v poole 1 000 ETH a 1 000 tokenov a Alice poskytne 500 ETH a 500 tokenov, potom má Alice nárok na 1/3 z poolu. Namiesto skladovania 500 ETH pre jej likviditu vám odporúčame uložiť, že vlastní 1/3 poolu. Počas swapov by sa táto hodnota nemala meniť (ani pri implementácii odmien pre podkytovateľov). Keď však poskytovateľ likvidity pridá alebo odoberie likviditu, táto hodnota (a všetky podiely ostatných poskytovateľov likvidity) by sa mala zodpovedajúcim spôsobom aktualizovať. Okrem toho, keďže Solidity nepodporuje desatinné miesta s pohyblivou čiarkou, odporúčame vám opraviť denomináciu (tj. 100, 1000 atď.), aby ste mohli percentá uložiť ako uint.
- **Chyby pri hraničných prípadoch**. Opäť, keďže Solidity nepodporuje desatinné čiarky, odporúčame, aby ste *pred delením vždy, keď je to možné, vykonali násobenie*. Vyhnete sa tak zaokrúhľovaniu delenia na 0 a následnému násobeniu, aby ste opäť dostali 0. Podobne pri typoch uint sa uistite, že ste *pred odčítaním vykonali sčítanie*, aby ste zabránili podtečeniu.
- **Schvaľovanie prenosov tokenov**. Aby mohla adresa tretej strany (napríklad kontrakt) odosielať alebo prijímať vaše tokeny vo vašom mene, musíte im najprv udeliť povolenie pomocou funkcie kontraktu tokenu `approve()`. Túto funkciu by musel iniciovať používateľ, nebudete môcť spustiť approve () zo samotného kontraktu. Preto sa uistite, že zavoláte túto funkciu v Javascripte pred volaním funkcie burzy. Podrobnosti o tejto funkcii možno nájsť v implementácii ERC20 Openzeppelin.
- **Odoslanie ETH do a z kontraktu**. Pre úspešný prevod ETH z používateľského účtu na kontrakt musí byť funkcia, ktorá prevod spracováva, označená ako *payable*. Je dôležité poznamenať, že jednoduché špecifikovanie argumentu vo funkcii kontraktu na určenie množstva ETH neprenesie ETH; skôr sa ETH prenáša cez parameter `msg.value`. **Aby ste sa vyhli desatinným miestam, pri vkladaní ETH do kontraktu**

použite `ethers.utils.parseUnits()` s jednotkami nastavenými na „wei“. Podobne, ak chcete poslať ETH z kontraktu na adresu používateľa, potom môžete vhodne použiť funkciu payable().

- **Iterácia cez kľúče v mape**. Solidity nepodporuje iteráciu pomocou kľúčov v mappingu. Preto definujeme `address[] private lp_providers`, aby ste uložili adresy poskytovateľov likvidity. Okrem toho nezabudnite, že polia Solidity automaticky „neposunú“ všetky prvky, keď sa odstráni hodnota v strede poľa. Preto poskytujeme pomocnú funkciu `removeLP()`, ktorá odstráni poskytovateľa likvidity z poľa a zároveň vyplní „medzeru“. Týmto sa zodpovedajúcim spôsobom aktualizuje `lp_providers.length`. Buďte opatrní pri volaní tejto funkcie počas iterácie cez pole, pretože je riskantné meniť dĺžku poľa počas iterácie, ak chcete dosiahnuť každý záznam.
- **Prístup k adrese kontraktu**. Adresu kontraktu v Solidity získate zavolaním `address(this).`
- **Prístup k množstvu tokenov a ETH v rámci kontraktu**. Aj keď môžete ručne vypočítať množstvo tokenov a ETH v kontrakte pri vykonávaní medzivýpočtov, k skutočným zostatkom máte prístup pomocou nasledujúcich funkcií:
  - ETH: `address(this).balance`
  - Token: `token.balanceOf(address(this))`

Dôrazne odporúčame použiť tieto skutočné hodnoty na výpočet *k* namiesto použitia `token_reserves` a `eth_reserves` a na dvojitú kontrolu svojej práce.

- **Asynchrónne funkcie Javascriptu**. V Javascripte je veľa prípadov, keď sú funkcie spúšťané asynchrónne. To znamená, že ak váš kód zavolá asynchrónnu funkciu, kód bude pokračovať v behu za týmto riadkom bez čakania na dokončenie vykonania tejto asynchrónnej funkcie. Predvolene, asynchrónne funkcie vracajú objekt Promise, ktorý určuje určitú hodnotu, ktorá sa vráti na konci vykonávania funkcie. Aby váš kód čakal na spustenie asynchrónnej funkcie a získanie výsledku, budete musieť použiť kľúčové slovo `await`.

Je dôležité poznamenať, že všetky volania Solidity funkcií z backendu budú asynchrónne. Ak teda chcete v súbore `exchange.js` zavolať funkciu z kontraktu a získať výstup, nezabudnite použiť kľúčové slovo `await`, ako napríklad `var num = await token_contract.function(args)`.

- `Bezpečnosť kontraktu`. Pamätajte, že keďže všetky nasadené kontrakty sú v blockchaine verejné, je dôležité, aby bol váš kontrakt bezpečný. Aj keď nebudeme

aktívne testovať bezpečnosť vašej implementácie, stále očakávame, že sa dočkáme určitých obranných opatrení prostredníctvom príkazov `require()` alebo `assert()`.

10. Sanity check

Aby sme otestovali vašu implementáciu, vytvorili sme dva programy na kontrolu, ktoré sa spúšťajú v závislosti od toho, či ste implementovali odmeny za likviditu. Tento stav skontrolujeme prečítaním hodnoty `swap_fee_numerator` v `exchange.sol`: ak je táto hodnota 0, potom predpokladáme, že ste neimplementovali swapové poplatky a odmeny za likviditu. Maximálne body viete získať aj bez odmien. Plne implementované odmeny za likviditu sú brané ako bonus do maximálnej výšky 3 bodov.

Ak chcete povoliť kontrolu sanity check, odkomentujte funkcie `setTimeout()` so `sanityCheck()` a obnovte webovú stránku. Snažili sme sa navrhnúť, aby sanityCheck fungoval správne aj po prvom načítaní, takže nie je potrebné znovu nasadzovať kontrakty a resetovať stav poolu zakaždým, keď chcete spustiť sanityCheck. Kvôli chybám zaokrúhľovania však môže nastať bod, v ktorom sanityCheck prejde, keď je výmenný kurz 1:1, ale nie s vaším aktuálnym stavom burzy. Vaše skóre sanityCheck pre tento projekt bude založené na počiatočnom stave poolu – to znamená, že výmenný kurz medzi tokenmi je 1:1 a v poole je 5 000 ETH a 5 000 tokenov.

11. Testovanie – BONUS za max 4 body

Ak sa rozhodnete spraviť si vlastné testy, tak je to viac než vítané a budeme to považovať za bonus. Pri každom testovaní je ale dôležité vyhodnotiť test coverage. Súčasťou odovzdania v tomto prípade musia byť aj vami napísané testy, ktoré pokryjú **aspoň 10% zdrojového kódu**. Nástroje na Solidity test coverage nájdete tu.

12. Dokumentácia

Súčasťou riešenia je aj dokumentácia, ktorá musí obsahovať najmä:

a) formálne náležitosti ako sú titulná strana, číslovanie strán;
b) odpovede na otázky z tejto sekcie;
c) popísané doimplementované časti kódu v skratke;
d) BONUS ak budete robiť - opis prostredia na testovanie a opis vybraných testov;
    a. sumár test coverage a použité nástroje;
e) BONUS - spracovanie security analýzy (stačí aj statická) – nástroje tu;
f) iné záležitosti, ktorými sa študenti chcú pochváliť;
g) záver a v ňom zhodnotenie vlastného príspevku a čo sme sa naučili.

V DesignDoc.txt máte otázky, na ktoré očakávame odpoveď v rámci dokumentácie:

- Vysvetlite, prečo pridávanie a odoberanie likvidity na vašej burze nezmení výmenný kurz.
- K bonusu - Vysvetlite svoju schému odmeňovania poskytovateľov likvidity a zdôvodnite rozhodnutia o dizajne, ktoré ste urobili. Ako spĺňa požiadavky na odmeny za likviditu uvedené v sekcii 7?
- Popíšte aspoň jednu metódu, ktorú ste použili na minimalizáciu spotreby gas pri kontrakte burzy. Prečo bola táto metóda efektívna?
- Voliteľná spätná väzba:
    - Koľko času ste strávili na zadaní?
    - Aká je jedna vec, ktorá by bola užitočná, keby ste ju vedeli predtým ako ste začali pracovať na zadaní 2?
    - Keby ste mohli zmeniť jednu vec v tomto zadaní, čo by to bolo?
    - Prosím pridajte nám akýkoľvek feedback alebo spätnú väzbu, ktorý máte na mysli alebo na srdci 😊.

Odovzdanie

Dokumentáciu a zdrojové kódy implementácie študent odovzdáva v elektronickom tvare do AISu do **04.05.2025 23:59** vo formáte *.zip alebo *.tar. Do zipu projektu nezabudnite zahrnúť nasledujúce súbory: exchange.js, exchange.sol, token.sol a dokumentácia. Ak ste pridali alebo zmenili akékoľvek iné súbory, nezabudnite zahrnúť aj tieto súbory. **Odovzdanie aspoň nejakej časti zadania je povinné!** Aj na toto zadanie je možné využiť Late days. V takom prípade ale pozor, aby obaja z dvojice mali ešte Late days voľné v takom množstve, ako chcete využiť. Všetky kódy prejdú kontrolou originality, v prípade vysokej percentuálnej zhody bude študentom začaté disciplinárne konanie.

Podmienkou je aj aktívne odprezentovanie programu študentom na cvičení podľa harmonogramu.

Môžete pracovať s 1 projektovým partnerom (veľkosť tímu = 2).

<u>Hodnotenie</u> (bez minima)

Celé riešenie - max. 20 bodov, z toho:

- max. 7 bodov za riešenie úlohy v exchange.sol – bude sa prihliadať čiastočne aj na celkové gas, pamäťové nároky, optimálnu práca s blockchainom;
- max. 2 bod za riešenie úlohy v token.sol – bude sa prihliadať čiastočne na kreativitu, primárne na funkčnosť;

- max. 7 bodov za riešenie úlohy v exchange.js – ako úspešne sa vám darí interagovať so smart kontraktom;
- max. 4 body za výslednú dokumentáciu – 1 bod formálna stránka, 1 bod odpovede na otázky, 2 body zvyšné časti.

Bonusy - dajú sa získať naviac k bodom za zadanie

- max. 4 body za vytvorené testy a celkový test coverage až do výšky 70-100% (za každých ďalších 20 % test coverage nad rámec povinných 10% pridelíme 1 bod, napr. celkový test coverage 30-49% = +1 bod; test coverage 50-69% = +2 body; test coverage 70-89% = +3 body; 90%+=+4 body);
- max. 2 body za vykonanie security analýzy – jej spracovanie, použité nástroje, identifikované hrozby a ich adresácia;
- max. 1 bod za pridanie nejakej zaujímavej funkcionality (po konzultácii s cvičiacim);
- max. 1 bod za využitie ďalších, do teraz nespomenutých, nástrojov zo zoznamu (po konzultácii s cvičiacim);
- max. 1 bod za prerobenie aplikácie z Hardhat frameworku do Foundry alebo Ape.