Лабораторная работа 2

Основы Solidity. Простой смарт-контракт

Теоретический блок

Помимо функции децентрализованного хранения данных о транзакциях, в блокчейне появились умные контракты, или смарт-контракты. Они представляют собой последовательность команд, описывающих условия сделки между участниками платформы. Смарт-контракты используются в блокчейн-среде как аналог коммерческих контрактов. Они являются самоисполняемыми и автоматически обеспечивают выполнение всех условий договора. К обязательным атрибутам «контракта» относят предмет договора, стороны, децентрализованную платформу и точно описанные условия.

Смарт-контракты могут быть как полностью автоматизированными, так и с копией на бумажном носителе. Также применяется технология, согласно которой в программный код перенесена лишь часть договора, а остальные условия закреплены на бумажном носителе. Это делается по той причине, что технология блокчейн пока не в полной мере может обеспечить создание сложных по своей структуре контрактов, и поэтому в основном автоматизируется обмен денежных средств между сторонами.

Широкое распространение смарт-контракты получили с развитием платформы Ethereum (Эфириум). Данная платформа обладает полнотой по Тьюрингу, что позволяет писать на ней сложные и совершенные «контракты» для решения максимально широкого диапазона задач. Платформа Ethereum написана с нуля, без использования кодовой базы биткоин. Биткоин не адаптирован в качестве среды для исполнения смарт-контрактов, так как он был разработан для использования только в качестве криптовалюты.

Для написания смарт-контрактов в среде Ethreum был специально разработан JavaScript-подобный язык программирования – Solidity. Solidity — один из четырех языков для EVM (три других: Serpent, LLL и Mutan) и, пожалуй, самый проработанный. Это статически типизированный JavaScript-подобный язык программирования. Среди прочих функций Solidity поддерживает наследование, библиотеки, события, перечисления и сложные пользовательские типы

Однако не смотря на то, что Solidity справляется с поставленными задачами и позволяет писать смарт-контракты практически любой сложности, он всё же не полностью реализован и некоторая функциональность в нем урезана. Например, функции не могут принимать в качестве параметров сложные типы, такие как массивы или перечисления. В настоящее время также ограничена поддержка типа string, так как длина слова в EVM равна 32 байтам. Следовательно, при компиляции могут возникнуть проблемы со строками, размер которых превышает 32 байта. Поэтому в официальной документации советуют всегда использовать тип bytes32 вместо string. Из-за ограничений EVM нельзя принимать или возвращать строки или массивы переменной длины. Единственным

решением является изначально ставить фиксированный большой размер массива, что очень усложняет разработку.

Несмотря на то, что технология блокчейн только набирает популярность, существует достаточно большое число инструментов, помогающих разработчикам выявить уязвимости на стадии разработки. Прежде всего это среды разработки и редакторы для создания смарт-контрактов на языке Solidity. Одной из наиболее популярных сред разработки является Open Remix IDE. Она удобна и проста в использовании. Данная среда разработки существует как в онлайн, так и в оффлайн версии. Она обеспечивает компиляцию кода, помогает в тестировании и отладке, предоставляет возможность аудита безопасности смарт-контрактов.

Краткий конспект:

Исходные файлы на языке Solidity имеют расширение .sol.

Исходные файлы должны быть аннотированы так называемой **version pragma**, чтобы предотвратить компиляцию в будущих версиях компилятора, которые могут привести к конфликтам.

Пример:

pragma solidity ^0.4.0;
pragma solidity >=0.4.0;

С помощью директивы *import* можно импортировать код из другого исходного файла.

- *import "./x" as x;* - импортирует global symbols из файла x, находящегося в той же директории, что и исходный файл.
- *import "x" as x;* - импортирует содержимое файла из директории импорта ().

Создание смарт-контрактов и их структура.

Смарт-контракт содержит переменные состояния (state variables), функции, модификаторы, события, структуры и перечисления.

Конструктор запускается во время создания контракта. Он постоянно хранит адрес того, кто создал контракт. Используется для первоначальной инициализации переменных и констант.

Переменные состояния – это глобальные переменные, которые постоянно хранятся в storage памяти контракта.

Под изменением состояния понимают:

1. Создание/изменение переменных.
2. Получение доступа к this.balance или <address>.balance.
3. Получение доступа к любому члену block, tx, msg (за исключение msg.sig и msg.data).

4. Вызов любой функции, которая не pure.
5. Использовать кода assembler, который содержит определенные операции.

Виртуальная машина Ethereum имеет три области памяти для хранения данных:
storage, memory и stack.

- memory - используем для хранения временных переменных.
- В storage хранятся переменные состояния, а также по умолчанию другие локальные переменные.
- stack - это рабочая память. + локальные переменные.
  stack - 1024 слота 32 байта.

Базовые типы Solidity:

Bool – позволяет хранить логические переменные (true, false). Оперрации производимые с данным типом: ! (логическое отрицание), && (конъюнкция), ||, (дизъюнкция) ==, !=.

Integer: int, int8, int256 - знаковые типы, uint, uint8, uitn256 - беззнаковые типы. Доступные операции: опреации сравнения (<=, <, ==, !=, >=, >) и арифметические операции (+, -, *, /, %, **).

Очень важно следить, чтобы не было переполнения, иначе переменная с типом Integer выдаст просто рандомное число.

Address: содержит значение в 20 байт (размер адреса Ethereum). С данным типом запрещены любые арифметические операции, однако разрешены операции сравнения. Тип address эквивалентен uint160, но в отличие от uint160 он предоставляет доступ к функциям получения баланса аккаунта, передачи валюты. Типа address также способствует повышению удобочитаемости, поскольку оно говорит о том, что сохраненное значение относится к адресу контракта.

bytes: Это специальный динамический массив плотно упакованных байтов в calldata - byte[], которые идут друг за другом.

String – это динамический массив плотно упакованных байтов в кодировке UTF-8. Вмесето строки рекомендуется использовать **bytes32**: Если заранее известна длина строки, рекомендуется использовать bytes1...bytes32, так как это значительно дешевле.

!!! bytes используется для необработанных байтовых данных произвольной длины, а string — для строковых (UTF-8) данных произвольной длины. bytes, в отличие от string, имеют функцию push.

Array:

- push – добавить элемент (someArray.push(newValue))
- length – узнать / переопределить длину
- инициализация массива: uint[] arr = new uint[] (1);

- многомерные массивы: uint[2][5] arr
- массив с динамическим размером: uint[] arr

Mapping – ассоциативный массив. Маппинги можно рассматривать как хэш-таблицы, в которых существуют всевозможные ключи,которым сопоставляются значения, изначально инициализированные нулем. Это не совсем точная аналогия, поскольку невозможно получить ни список всех ключей, ни список всех значений. Поэтому нужно лучше использовать маппинг там, где это нужно. Часто вместе с маппингом создается дополнительный массив для хранения списка ключей для подсчета количества элементов.
Маппинг может быть многоуровневым.

mapping (_KeyType => _ValueType)

Структуры (**Structs**) позволяют в Solidity создавать пользовательские типы данных. Они могут использоваться в маппингах или в массивах, и сами содержать массивы и маппинг. Структура не может содержать элемент своего типа.

Enum (перечисления) - Перечисление объявляется с помощью ключевого слова enum. Это способ создания пользовательского типа в Solidity. Они явно конвертируются в и из всех целых типов, но неявное преобразование не допускается. Явные преобразования проверяют диапазоны значений во время выполнения, а ошибка вызывает исключение. Для перечислений обязательно наличие хотя бы одного члена.

enum SomeData {DEFAULT,ONE,TWO}

Ключевые слова в Solidity

Эфир делится на следующие единицы: **wei, finney, szabo**. Сам эфир представлен в Solidity как **ether**.

!!! Время в Solidity:

- Нельзя получить текущее время.
- **now** - время текущего блока. Не стоит полагаться на **block.timestamp**, **now** и **block.blockhash**. Как временная метка, так и блок-хэш могут в некоторой степени быть подвержены влиянию майнеров. Текущая временная метка блока должна быть строго больше, чем метка времени последнего блока, но единственная гарантия заключается в том, что он будет находиться где-то между отметками времени двух последовательных блоков в канонической цепочке.
- В Solidity представлены следующие единицы измерения времени: seconds, minutes, hours, days, weeks и years.

Особые переменные и функции

- Глобальная переменная **this** указывает на текущий контракт, явно конвертируемый в адрес.
- Глобальная перемен **msg** — это переменная, которая содержит свойства, предоставляющие доступ к таким данным блокчейна, как отправитель транзакции, данные, переданные с помощью транзакции, первые четыре байта calldata, количество wei отправленных с транзакцией.

msg.sender — это адрес отправителя текущего внешнего (external) вызова функции.

tx.origin - отправитель транзакции, **msg.sender** - отправитель текущего вызова.

В простой цепочке вызовов A-> B-> C-> D внутри D msg.sender будет C, а tx.origin будет A.

Обработка ошибок

- assert(bool condition): выбрасывает исключение, если условие не выполняется.
- require(bool condition): выбрасывает исключение, если условие не выполняется.
- require(bool condition, string message): выбрасывает исключение, если условие не выполняется, также появляется сообщение об ошибке.
- revert(): прекращает выполнение и возвращает изменения состояния.
- revert(string reason): прекращает выполнение и возвращает изменения состояния,также появляется сообщение об ошибке.

Ознакомиться с полной документацией по Solidity можно здесь:
https://solidity.readthedocs.io/en/v0.5.12/index.html

Практический блок

1. ВНИМЕТЕЛЬНО изучить раздел «Introduction to Smart Contracts»

https://solidity.readthedocs.io/en/v0.5.12/introduction-to-smart-contracts.html#a-simple-smart-contract

!!! Если в тексте документации встречаются примеры, обращайте на них внимание!

Для работы со смарт-контрактами мы будем использовать среду разработки Open Remix IDE: https://remix.ethereum.org/

2. Написать смарт-контракт для аукциона. (требования описаны ниже)

Смарт-контракт Аукцион

Версия Solidity - 0.4.25
Смарт-контракт должен содержать функции:

- начать новый аукцион и установить минимальную ставку

- сделать ставку - Каждая новая ставка должна превышать предыдущую. Аукцион заканчивается, когда больше нет желающих превысить последнюю ставку
- завершить аукцион: объявляем победителя, высылаем обратно все остальные ставки.

Доп. требования:

- Только владелец смарт-контракта может создавать новые аукционы
- Одновременно может работать два аукциона
- В смарт-контракте должны быть предусмотрены все варианты проверок.
- !Имя смарт-контракта должно содержать фамилию студента.

3. Необходимо скомпилировать смарт-контракт с помощью IDE Remix. ABI предоставить в файле с отчётом

Контрольные вопросы

1. Перечислите (кратко!) основные преимущества смарт-контрактов.
2. Назовите особенности законодательства в отношении смарт-контрактов и криптовалюты в Беларуси (декрет о ПВТ 2.0)
3. Что такое ABI смарт-контракта?
4. Что содержит в себе файл, сгенерированный компилятором Solidity?
5. Перечислите основные составляющие смарт-контракта.
6. Что такое переменные состояния? Есть ли разница в стоимости между функций создания новой переменной состояния или ее изменением?
7. Есть ли возможность в Solidity получить текущее время? Можно ли полагаться на встроенную функцию now?
8. В чем отличие string от bytes?

Задание

1) Изучить теоретический материал

2) Выполнить практическую часть лабораторной работы

3) Ответить на контрольные вопросы

4) Оформить отчёт

5) прикрепить файл смарт-контракта с расширением .sol

Защита лабораторной работы 1 (часть 2): отчёт и файл с кодом смарт-контракта должны быть отправлены на портал.

Общие требования к отчёту

Отчёт должен содержать следующие элементы:

1) номер и название лабораторной работы,

2) ФИО студента, группу и курс

3) выполненные задания практического блока: ABI смарт-контракта и скриншот с IDE Remix

4) краткие ответы на контрольные вопросы.

В случае обнаружения плагиата (в том числе и в ответах на контрольные вопросы) отчёт не будет принят преподавателем!