Лабораторна робота 3
ТИПИ ДАНИХ ТА СТРУКТУРИ В SOLIDITY

Теоретичні відомості

Вступ

Для створення і реалізації смарт-контрактів у блокчейні Ефіріум використовується мова Solidity –об'єктно-орієнтована мова високого рівня для реалізації смарт-контрактів.

Мова була запропонована в серпні 2014 року Гейвіном Вудом. Надалі розробка мови була виконана під керівництвом Крістіана Райтвизнера командою Solidity в рамках проекту Ethereum. Це одна з чотирьох мов (три інші: Serpent, LLL і Mutan), спроектованих для трансляції в байт-код віртуальної машини Ethereum.

На Solidity впливали C++, Python та JavaScript, і він створений з орієнтацією на віртуальну машину Ethereum (EVM). Solidity має статичну типізацію, підтримує успадкування, бібліотеки та складні визначені користувачем типи серед інших функцій.

Завдяки Solidity ви можете створювати контракти для таких цілей, як голосування, краудфандинг, сліпі аукціони та гаманці з багатьма підписами тощо.

Solidity є повноцінною мовою програмування, яку ще називають тюрінг-повною мовою. Вона дозволяє нам кодити за допомогою усього спектру інструментів, які є у більшості мов програмування.

Смарт-контракти – це програми, які регулюють поведінку рахунків (аккаунтів) у блокчейні Ethereum.

Контракти в Solidity подібні класам в об'єктно-орієнтованих мовах програмування. Кожен контракт може містити оголошення глобальних змінних, функцій, модифікаторів функцій, подій, структур і перерахувань.

Глобальні змінні

Глобальні змінні – це значення, які постійно зберігаються в сховищі контракту.

```
pragma solidity ^ 0.4.0; contract SimpleStorage { uint storedData; // Глобальна змінна
}
```

Функції

Функції – це виконувані одиниці коду всередині контракту.

```
pragma solidity ^ 0.4.0;
contract SimpleAuction {
   function bid () public payable { // Функція
   }
}
```

Виклик функції може бути внутрішнім (internal) або зовнішнім (external).

Функції можуть мати різні рівні видимості по відношенню до інших контрактів.

Модифікатори функцій

Модифікатори функцій можна використовувати, щоб змінити семантику функцій декларативним способом.

```
pragma solidity ^ 0.4.11; contract Purchase
{    address public seller;    modifier
onlySeller () {// Модифікатор        require
(msg.sender == seller);
        _;
   }
   function abort () public onlySeller {// Використання модифікатора
            // ...
   }
}
```

Події

Події – це зручні інтерфейси з можливістю ведення логу EVM.

```
pragma solidity ^ 0.4.0; contract SimpleAuction {     event
HighestBidIncreased (address bidder, uint amount); // Подія     function
bid () public payable {
    // ...
      HighestBidIncreased (msg.sender, msg.value); // Виклик події
   }
}
```

Тип "структура"

Структури – це типи даних, які можуть групувати кілька змінних.

```
pragma solidity ^ 0.4.0;
contract Ballot {     struct
Voter {// Структура         uint
weight;         bool voted;
address delegate;         uint
vote;
   }
}
```

Тип "перерахування" (Enum)

Перерахування можуть використовуватися для створення настроюваних типів з кінцевим числом значень.

```
pragma solidity ^ 0.4.0; contract Purchase {     enum
State {Created, Locked, Inactive} // Enum
}
```

Операции с массивами

Append(src array, val someType) array

Функция вставляетв src значение val любого типа и возвращает результирующий массив

- *src* - исходный массив
- *val* - значение, которое необходимо добавить в массив

```
var list array
list = Append(list, "new_val")
```

Join(in array, sep string) string

Функция объединяет элементы массива *in* в строку с указанным разделителем *sep*.

- *in* - имя массива типа *array*, элементы которого необходимо объединить,
- *sep* - строка-разделитель.

```
var val string, myarr array
myarr[0] = "first"
myarr[1] = 10
val = Join(myarr, ",")
```

Split(in string, sep string) array

Функция возвращает массив, полученный из элементов строки *in*, при ее разбивании в соответствии с разделителем *sep*.

- *in* - исходная строка,
- *sep* - строка-разделитель.

```
var myarr array
myarr = Split("first,second,third", ",")
```

Len(val array) int

Функция возвращает количество элементов в указанном массиве.

- *val* - массив типа *array*.

```
if Len(mylist) == 0 {
  ...
}
```

Row(list array) map

Функция возвращает первый ассоциативный массив *map* из массива *list*. Если список *list* пустой, то результат вернет пустой *map*. Используется преимущественно с функцией DBFind, в этом случае параметр *list* не указывается.

- *list* - массив map, возвращаемый функцией **DBFind**.

```
var ret map
ret = DBFind("contracts").Columns("id,value").WhereId(10).Row()
```

```
Println(ret)
```

One(list array, column string) string

Функция возвращает значение ключа *column* из первого ассациативного массива в массиве *list*. Если список *list* пустой, то возвращается nil. Используется преимущественно с функцией DBFind, в этом случае параметр *list* не указывается.

- *list* - массив map, возвращаемый функцией **DBFind**,
- *column* - имя возвращаемого ключа.

```
var ret string
ret = DBFind("contracts").Columns("id,value").WhereId(10).One("value")
if ret != nil {
  Println(ret)
}
```

GetMapKeys(val map) array

Функция возвращает массив ключей из ассациативного массива *val*.

- *val* - массив map.

```
var val map
var arr array
val["k1"] = "v1"
val["k2"] = "v2"
arr = GetMapKeys(val)
```

SortedKeys(val map) array

Функция возвращает отсортированный массив ключей из ассациативного массива *val*.

- *val* - массив map.

```
var val map
var arr array
val["k1"] = "v1"
val["k2"] = "v2"
arr = SortedKeys(val)
```

Індивідуальне завдання

1. Написати контракт, який повинен реалізовувати інтерфейс IDataTypes.sol

a) для кожної функції, крім getBigUint(), повинна бути створена змінна на рівні контракту. Значення змінної задати при об'явленні змінної;

b) значення для змінною типу String має бути "Hello World!";

c) функція getBigUint() повинна мати дві локальні змінні v1 та v2 із значеннями 1 та 2 відповідно. Ця функція повинна повернути число більше 1000000. Дозволяється використовувати лише арифметичні оператори та лише по одному разу кожний;

d) функція getBigUint() є додатковим завданням.

2. Написати контракт, який повинен реалізовувати інтерфейс IStudentRegistrar.sol

a) функція setNewStudent() повинна встановлювати передану інформацію (структура в параметрі функції) для адреси, яка відправила транзакцію;

b) функція getStudent() повинна повертати інформацію для конкретної адреси;

c) функція getMyInfo() повинна повертати інформацію для відправника транзакції;

d) для зберігання даних використовувати mapping.

3. Посилання на валідатора до цієї лабораторної роботи Lab3Validator