Lab 2 – Introducción a *Remix*

Remix es un entorno de programación creado por la comunidad de desarrolladores de Ethereum. Está disponible online en `https://remix.ethereum.org` y también puedes descartar una versión de escritorio. En la página principal de Remix existe un enlace a toda la documentación del sistema.

Este entorno de desarrollo tiene los siguientes componentes que hemos visto en la clase de laboratorio:

- File explorer.
- Solidity compiler.
- Deployment and transaction execution environment.
- Debugger.
- Plugin manager.
- Settings.

Esta primera sesión introduce el uso de Remix y algunos elementos básicos del lenguaje de programación Solidity. Comienza creando un fichero de texto con cualquier editor de texto para escribir en él las respuestas a los ejercicios de este documento. No olvides poner los nombres de los componentes del grupo al principio del fichero. Debes entregar este fichero en la tarea correspondiente del Campus Virtual.

1. Mi primer programa Solidity

Inicia el entorno web de Remix y crea un nuevo fichero fuente `hello.sol` con el siguiente contenido:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.8.0;

contract hello {
    event Print(string message);

    function helloWorld() public {
        emit Print("Hello, World!");
    }
}
```

Ejercicio 1. *Compila este program, despliégalo y ejecuta una transacción que invoca la función* `helloWorld`*. Después de ejecutarlo, despliega los resultados de ejecución en el panel inferior de la pantalla e inspecciona los resultados:* transaction cost, execution cost, logs, value*. Escribe estos resultados en tu fichero de respuestas de esta sesión.*

Este contrato mínimo utiliza un **evento** para mostrar un mensaje. Esta es la forma de publicar datos desde un contrato que se ejecuta en la EVM a una aplicación externa. Para comunicar resultados a otras funciones de este u otros contratos puedes utilizar la instrucción **`return`**, igual que en una función en C o un método en Java.

2. Jugando con Solidity y Remix

En Solidity se puede programar de forma muy similar a cualquier otro lenguaje de programación. Su sintaxis es muy parecida a otros lenguajes imperativos como C, Java o Javascript. Solidity tiene varios tipos de datos diferentes a los de estos lenguajes, pero en esta primera sesión solo utilizaremos **`string`** y **`uint`** (para representar enteros sin signo).

Ejercicio 2. *Añade al contrato* `hello` *otra función pública para calcular el factorial de un número. Utiliza el siguiente prototipo de función:*

```
function factorial(uint n) public pure returns (uint)
```

Utiliza el tipo de datos **`uint`** *para las variables locales y devuelve el resultado con la instrucción* **`return`***. Compila el programa, vuelve a desplegarlo y ejecuta una transacción que invoque la función* `factorial` *para calcular el factorial de 50. Puedes observar que las variables enteras pueden manejar números de hasta 256 bits.*

Añade el código del contrato y el resultado de calcular el factorial de 50 al fichero de respuestas.

3. Enviando ether a un contrato

Cuando se despliega un contrato se le asigna una cuenta de contrato con un saldo inicial de 0 ETH. Se puede enviar ether a un contrato mediante una transacción que invoque a una función del contrato. Para ello, hay que adaptar el código del contrato para que pueda tratar el envío:

- Para que una función acepte ether cuando sea invocada, debe anotarse mediante la palabra **`payable`**.
- Se puede consultar el saldo actual de la cuenta del contrato con la expresión `address(this).balance`.
- Se puede enviar dinero al emisor de la transacción utilizando

  ```
  payable(msg.sender).transfer(amount);
  ```

  donde `amount` es la cantidad a enviar en Wei.

Ejercicio 3. *En este ejercicio vamos a implementar una hucha minimalista. Crea un nuevo contrato* `PiggyBank0` *en un fichero llamado* `piggyBank0.sol`*. Este contrato debe tener tres funciones:*

- ***function*** `deposit()` ***external payable***
  Crea esta función con un cuerpo vacío para poder aceptar ether enviado por una cuenta externa.

- ***function*** `withdraw(` ***uint*** `amountInWei)` ***external***
  Esta función sirve para retirar dinero de la hucha. Primero debe comprobar si la cuenta tiene suficiente ether para enviárselo al emisor de la transacción. Si hay suficiente ether, debe enviar la cantidad solicitada al emisor, o debe emitir un mensaje de error con un evento en caso contrario.

- ***function*** `getBalance()` ***external view returns*** `(` ***uint*** `)`
  *Esta función debe devolver como resultado de la llamada el saldo (*balance*) de la cuenta asociada al contrato.*

Prueba el contrato de la siguiente forma:

1. *Despliega este contrato y deposita 3 ETH. Utiliza el campo de texto "Value" del panel de la izquierda para especificar la cantidad a enviar justo antes de invocar a la función* `deposit`*.*

2. *Después cambia la cuenta EOA que realiza la transacción utilizando la lista "Account" que aparece en el panel izquierdo, seleccionando una cuenta que tenga 100 ETH de saldo. Intenta retirar 5 ETH y comprueba que se ha emitido el evento y que no se ha retirado ninguna cantidad de ether. Verifica que el saldo de la cuenta EOA solo ha decrementado el coste de la transacción.*

3. *Finalmente, Cambia de nuevo la cuenta EOA a otra cuenta que tenga 100 ETH de saldo. Retira 1 Szabo (*10^{12} *wei) y comprueba el saldo del contrato invocando a* `getBalance` *y el saldo de la cuenta EOA desde la que has realizado la retirada de dinero. Verifica que el saldo de la cuenta EOA es 100 ETH más 1 Szabo menos el coste de la transacción.*

Documenta estos tests en tu fichero de respuestas detallando para cada test y función invocada lo siguiente: transaction cost, execution cost, decoded output, logs *y* value. *Incluye también el saldo de las cuentas EOA que has utilizado.*

El contrato `PiggyBank0` no es muy "smart", pues permite retirar dinero desde cualquier cuenta de usuario. En otra sesión de laboratorio veremos la forma de corregir este problema y comprobar que el creador del contrato es el único que puede retirar dinero de la hucha.