Desarrollo Blockchain Ethereum con Solidity

Módulo 3 - Tipos de contratos y funciones

Tipos de dato

Tipos de visibilidad en las funciones

Existen distintos tipos de visibilidad en las funciones que determinan desde dónde podemos acceder a la información de las mismas:

- **Public**: Es visible desde todo ámbito.
- **Private**: es visible sólo desde mi contrato.
- **Internal**: es visible sólo desde mi contrato y desde los contratos derivados.
- **External**: es visible sólo desde fuera mi contrato (es decir, desde llamadas externas).

Librerías

- Similares a los contratos, pero se implementan una sola vez en la blockchain.
- El código que se ejecuta utiliza el contexto del contrato “llamador”, es decir que es similar a la instrucción **delegatecall.**
- No están pensadas para guardar un estado, por lo cual trabaja siempre sobre el estado que lo llama pero no guarda información en su propio estado.
- En cierta forma son como “contratos base” para nuestros contratos, o bien proveedores de utilidad.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.9.0;

struct DatoListado {
    mapping(uint => bool) flags;
}

library ListadoNumerico {
    function agregar(DatoListado storage listado, uint valor)
        public
        returns (bool)
    {
        if (listado.flags[valor])
            return false; // Ya lo contiene
        listado.flags[valor] = true;
        return true;
    }

    function quitar(DatoListado storage listado, uint valor)
        public
        returns (bool)
    {
        if (!listado.flags[valor])
            return false; // No lo contiene
        listado.flags[valor] = false;
        return true;
    }

    function contiene(DatoListado storage listado, uint valor)
        public
        view
        returns (bool)
    {
        return listado.flags[valor];
    }
}
```

Sobrecarga de funciones

Al igual que en otros lenguajes, es posible definir dos funciones con el mismo nombre pero distinto tipo o cantidad de variables.

Hay que tener cuidado a la hora de sobrecargar funciones ya que existen algunos errores comunes, como tener dos funciones con el mismo encabezado pero con parámetros enteros de distinto tamaño, en ese caso se llamará siempre a la primera definida ya que no puede determinar con claridad a cuál de las dos quiere llamar.

```
function sumar(uint a, uint b) public pure returns(uint) {
    return a + b;
}

function sumar(uint a, uint b, uint c) public pure returns(uint) {
    return a + b + c;
}
```

Tipos de almacenamiento

- **Storage**: persiste entre funciones y transacciones. Cuesta gas tanto leer como escribir por lo cual es necesario minimizar el uso de este almacenamiento.
- **Memory**: se obtiene una nueva instancia en cada llamada. Se utiliza cuando no deseamos persistir la información que estamos leyendo/escribiendo.
- **Stack**: es el almacenamiento por defecto de la EVM, donde se realiza todo el cómputo. Si bien es accesible, los desarrolladores no suelen utilizar este medio de almacenamiento dentro de los contratos.
- Cuando escribimos parámetros, el compilador nos sugiere utilizar a veces la palabra **calldata**. Esto referencia al lugar donde se alojan los parámetros los cuales sirven para lectura, pero no se pueden modificar o escribir.

Tipos de almacenamiento

Recordemos que todos los datos que almacenamos en la red tienen un gasto asociado. Almacenar en el “storage” es más costoso que en memoria, pero no por eso utilizar la memoria implica un costo bajo.

Por lo tanto, es una buena práctica elegir memory sobre storage para optimizar el uso del gas en nuestro contrato.

¡Muchas gracias!

¡Sigamos trabajando!