**Practica 7**

**Ej 1:** Función maxMinMemory programada en Yul para memoria con la función en el bloque assembly fmaxmin:

```
function maxMinMemory(uint[] memory arr) public pure returns (uint maxmin){    infinite gas
    assembly {

        function fmaxmin(array_pointer) -> maxVal, minVal {

            let len := mload(array_pointer) // cargamos de memoria la longitud del array
            let data := add(array_pointer, 0x20) // avanzamos a la posicion de memoria del primer elemento
            maxVal := mload(data)
            minVal := mload(data)
            let i := 1


            for {} lt(i, len) { i := add(i, 1) } {
                let elem := mload(add(data, mul(i, 0x20))) // cargamos el siguiente elemento
                if gt(elem, maxVal) { // si es mayor el elemento q el maximo -> es el maximo
                    maxVal := elem
                }
                if lt(elem, minVal) { // si es menor el elemento que el minimo -> es el minimo
                    minVal := elem
                }
            }
        }

        let maxVal, minVal := fmaxmin(arr)
        maxmin := sub(maxVal, minVal)
    }
```

```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: lab7.maxMinMemory(uint256[]) data: 0x8fb...00005

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                      lab7.maxMinMemory(uint256[]) 0x7EF2e0048f5bAeDe046f6BF797943daF4ED8CB47

execution cost          1286 gas (Cost only applies when called by a contract)

input                   0x8fb...00005

decoded input           {
                            "uint256[] arr": [
                                "1",
                                "2",
                                "3",
                                "4",
                                "5"
                            ]
                        }

decoded output          {
                            "0": "uint256: maxmin 4"
                        }

logs                    []
```

**Ej 2:** Dado el contrato lab6ex6 programamos la función maxMinStorage que realice el mismo cálculo que el apartado anterior para un array arr en storage. El contrato tiene la variable uint[] arr;

```
function maxMinStorage() public view returns (uint maxmin) {

    uint256 s;

    assembly{
        s := arr.slot
    }

    bytes32 loc = keccak256(abi.encode(s));

    assembly {
        //no se si es asi, muy parecido al anterior
        function fmaxmin(slot, n) -> maxVal, minVal {
            let len := sload(n)
            maxVal := sload(slot)
            minVal := sload(slot)

            for {let i := 1} lt(i,len) { i := add(i,1) } {
                let elem := sload(add(slot, i))
                if gt(elem, maxVal) { // si es mayor el elemento q el maximo -> es el maximo
                    maxVal := elem
                }
                if lt(elem, minVal) { // si es menor el elemento que el minimo -> es el minimo
                    minVal := elem
                }
            }
        }

        let maxVal, minVal := fmaxmin(loc, arr.slot)
        maxmin := sub(maxVal, minVal)
    }
}
```

```
[vm] from: 0x5B3...eddC4 to: lab6ex6.generate(uint256) 0xd91...39138 value: 0 wei data: 0x4a7...00064 logs: 0 hash: 0xe08...fc92a

status                  0x1 Transaction mined and execution succeed
transaction hash        0xe080284bdf58b7cb33baa367fb78e685dac2432d9897c3cadc1e119452bfc92a
block hash              0x3dcd19da727c1354894aca11314d8c0523def9ba69dbcdde5c044f2fd8cd6c29
block number            2
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      lab6ex6.generate(uint256) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas                     2644719 gas
transaction cost        2299755 gas
execution cost          2278551 gas
input                   0x4a7...00064
decoded input           {
                            "uint256 n": "100"
                        }
decoded output          {}
logs                    []
```

```
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: lab6ex6.maxMinStorage() data: 0xb99...9fafd

from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      lab6ex6.maxMinStorage() 0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost          224027 gas (Cost only applies when called by a contract)
input                   0xb99...9fafd
decoded input           {}
decoded output          {
                            "0": "uint256: maxmin 240"
                        }
logs                    []
```

**Ej 3:** Implementación simple de maxMinStorage() escrita en Solidity, y creamos un array de 100 elementos para el contrato lab6ex6 y para este lab7Simple para comparar los consumos de gas.

```
function maxMinStorage() public view returns (uint maxmin) {    infinite gas
    uint max = arr[0];
    uint min = arr[0];
    uint elem;

    for(uint i = 1; i < arr.length; i++){
        elem = arr[i];

        if(elem > max) max = elem;
        if(elem < min) min = elem;
    }

    maxmin = max - min;
}
```

generamos array de 100 elems para el contrato lab7Simple


Ejecutamos la función maxMinStorage:

gas de maxMinStorage para array de 100 elementos en contrato lab6ex6: 224017 gas

gas de maxMinStorage para array de 100 elementos en contrato lab7Simple: 252924 gas

Como podemos apreciar existe una diferencia de gas consumido al hacerlo mediante Yul respecto a hacerlo mediante Solidity simple, esto se debe a que Yul permite escribir en lenguaje a más bajo nivel, estando más cerca del nivel de la máquina virtual Ethereum y esto permite tener un menor consumo de gas respecto a la implementación simple en Solidity.