SOLIDITY – HOMEWORK 2

Exercise 1 - Function Encoding (1,5 + 0,5 points)

From the example on slide 10 and 11 of Lecture 3, what would be the output for the second example (HelloWorld(uint[], bool))? You can calculate the function selector with the provided algorithm on slide 11. For the memory location, you can make it up but remember to observe left and right zero padding for all inputs. (1,5 points)

After deploying the function in Remix IDE, here is the ABI-encoded output:

Function Selector: The first 4 bytes correspond to the function signature hash (0x23ffcc4c), calculated using keccak256("HelloWorld(uint[],bool)").

Memory Location of uint[]: This represents where the array is located, here it's 0x40, or 64 in decimal, which is the location of the array in memory.

Array Length: The length of the array is 0x20 (32 bytes) due to the two elements in the array, with each element takes 32 bytes.

The array has two elements, which are:

- First element (1993): 0x00000000000000000000000000000000000000000000000000000000000007c9
- Second element (1994): 0x00000000000000000000000000000000000000000000000000000000000007ca

Boolean Value: "true" is encoded as 0x01 (in 32 bytes) and represented as 0x0000000000000000000000000000000000000000000000000000000000000001.

So the final output is : Function selector + memory location of unit[] + bool + length of array + values in array

What would the encodePacked function call give you? (0,5 points)

Function Selector: The first 4 bytes correspond to the function signature hash (0x23ffcc4c), calculated using keccak256("HelloWorld(uint[],bool)").

The array elements (1993 and 1994) are encoded directly as:

- 1993 : 0x00000000000000000000000000000000000000000000000000000000000007c9

- 1994 : 0x00000000000000000000000000000000000000000000000000000000000007ca

Boolean Value: “true” is encoded as 0x01 (in 32 bytes) and represented as 0x0000000000000000000000000000000000000000000000000000000000000001.

So the final output is : Function selector + bool + values in array

Exercise 2 - An upgradeable Coinflip (5 + 5 + 2 points)

Part C - Upgrade mechanism (2 points)

There are other types of contract mechanisms besides the 2 presented in class. Chose a more different mechanism, briefly describe how it works and how it compares to the transparent and UUPS models.

Another contract upgradeability mechanism besides the Transparent Proxy and UUPS Proxy models is the Beacon Proxy pattern. In the Beacon Proxy model, the upgradeable contract pattern works by separating the logic contract from the beacon contract (which stores the address of the current implementation).

The key idea is that the beacon contract holds the address of the contract that should be called for logic execution (i.e., the implementation contract). This allows you to change the logic contract without modifying the proxies, as they delegate calls to the beacon for the address of the implementation contract. The beacon can be updated to point to a new contract address, effectively upgrading the logic of multiple proxies at once.

In contrast to the Transparent and UUPS models, the Beacon Proxy pattern decouples the proxy contract and the implementation contract entirely by introducing a Beacon that stores the address of the implementation.