Ethereum
Blockchain Development
Week 16

Introduction

The intention of this lab is to look at general programming in solidity. All the exercises are completed in Remix IDE.

Code Completion

Writing code in a new language can be a steep learning curve. The approach here is to provide some code with underscores (_) that you are required to complete. These underscores are there to help you. By completing these exercises you will be improving your skills and knowledge of Solidity. Coding requires practice, so you will need to practice these and other exercises outside this lesson as independent study.

Each exercise starts on a new page. The red numbers in the right-hand margin are estimated minutes you should spend on each exercise.

Code Completion

```
1  _/____-License-__________: MIT
2  ______ solidity ______;
3  ________ ex1{
4      ____ Gender {male, female, nonBinary}
5      ______ Person{
6          string firstName;
7          ______ lastName;
8          _____ age;
9          Gender gender;
10     }
11     Person person;
12     ________ setPerson(string memory _firstName, string memory _lastName, uint8 _age, Gender
         _gender) public {
13         ______.firstName    = _firstName;
14         ______.lastName     = _lastName;
15         ______.age          = _age;
16         ______.gender       = _gender;
17     }
18     function getPerson() ______ ____ returns(_erson ______){
19         return person;
20     }
21     function genderQ() public ____ _______(_ender){
22         return ______.gender;
23     }
24 }
```

Figure 1: Structure for `ex1.sol`, type this in a Ethereum contract

1 Structures

5–10

Complete the contract that contains two functions that store and display the structure in Fig. 1.

Consider why it is a bad idea to implement this on a blockchain?

Code Completion

```
//_____License___________: GPL2.0
______ solidity ^0.8.0;
________ ex2{
    _____ ________ size = 100;
    uint[____] array = [10, 20, 30, 40];

    ________ populate() ______ {
        for(uint i=0; i<size; i++)
            array[i]=i;
    }
    function sumArray() _______ ____ _______(uint){
        ____ sum=0;
        for(____ i=__ i<____; i__)
            sum+=_____[i];
        return ___;
    }
}
```

Figure 2: Structure for `ex2.sol`, type this in a Ethereum contract

TX Cost	Parameters
005621930015741404	100 iterations; setter; using 256 bits
000567977501590337	100 iterations; setter; using 16 bits and conversions
000119852500335587	10 iterations; setter; using 16 bits and conversions

Table 1: Table for optimising transaction costs, Gas price: 00000002500000007 Wei

2 Arrays

15–20

Complete the contract in Fig. 2 that that stores numbers 0,..,99 in an array as a state variable.

Complete the function that populates the values in the array.

Write a function that returns the sum of the all the elements in the array.

Whats the problem? With an array of hundred the transaction cost of setting the array was: 2248772 Wei. Does the reduction in array size reduce the transaction cost? Repeat the problem with an array size of 5, 10 and 100 and record the cost for setting the array. Are there other ways of optimising the cost? Table 1 shows how optimising parameters can have an effect on transaction cost, it also demonstrates the reduction in cost caused by a reduction in iterations.

3 Iteration and Selection

20–25

Add a function to the previous exercise in §2 that only adds even numbers. Use a for loop and the continue function to complete this.

3.1 Break

Add a function to calculate the sum of the first n elements of a fixed array of size 20 containing unsigned integers in the range of 0,..,255. There should be an additional function to enter a value n, which is stored as a local variable and has to be less than 20, the size of the fixed array. Use a while loop and a break command to complete this.

3.2 Event

Write an event called `notifySum` that adds an integer to the blockchain. Use the `emit` keyword to record the sum in both the functions above using this event. Test and evaluate both your functions.

Code Completion

```
1  //SPDX-_______-Identifier: ___
2  pragma ________ ^0.8.0;
3  ________ ex4{
4      uint8 ________ size = 100;
5      _______ (uint => address) ______ addressMapping;
6      _______[size] ______ addressArray;
7      uint maxIndex;
8
9      ________ addAddressBookEntry( _______ _entry) ______ {
10         ________+=1;
11         addressMapping[________]=_entry;
12     }
13
14     ________ convertToArray() ______ {
15         for (uint i=1; i<=________; i++)
16             addressArray[i-1] = addressMapping[i];
17     }
18 }
```

Figure 3: Listing for `ex4.sol`, mapping for an Ethereum contract

4 Mappings

20–25

This section looks at mappings and conversions to arrays. Create a new contract. Examine and complete the code in Fig.3. How much more efficient is a mapping than an array?

Unit	Calculation	Seconds
Day	24*60*60	86,400
Hour	60 * 60	3,600

Table 2: Seconds in a Day and Hour

Code Completion

```
__SPDX-_______-Identifier:MIT
pragma ________ _0.8._;

________ ex5_
    uint date = block.timestamp;
    uint remaining;
    event display(____);
    ________ secondsToNextBirthday(uint bday) ______ {
        remaining = bday - date;
        emit _______(remaining);
    _
_
```

Figure 4: Listing for `ex5.sol`, calculating date differences in an Ethereum contract

5 Dates

5–10

Dates in solidity are stored in unix timestamps, which means they can be store as integers. Unix timestamps are the number of seconds elapsed since 01/01/1970.

Table 2 is a brief outline of the number of seconds in each time unit.

Create a new contract and complete the code listing in Fig.4. To calculate your next birthday in Unix timestamp, go to time.is and enter your next birthday. Then execute the function.

6 Card Game

25–30

This is a simple game of guessing the card selected. The rules of the game are simple, and as follows:

1. Player One selects a card, that is stored in a state variable: `playerOneCard`.
2. Player Two selects a card, that is stored in a state variable: `playerTwoCard`.
3. The objective is for Player Two to guess the card Player One has selected.
4. If the card select by Player Two matches the card selected by Player One, then this is the end of the game.
5. If the two cards selected don't match then Player Two selects another card and the process is repeated - go to step 2.

Follow the steps below to complete the task above:

1. Create a new file and contract.
2. In the contract have two enum types for the `Suit` and the `Value`.
3. In the contract have a structure to represent all cards in the deck, `Card`.
4. Create two public state variables: `playerOneCard` and `playerTwoCard`, that allow for a `Card` to be stored.
5. Create two functions: `selectPlayerOneCard` and `selectPlayerTwoCard`, that facilitate the selection of each card to `playerOne` and `playerTwo`, respectively.
6. Create one function that tests if the two cards are equal.

7 Reading

Read Chapter 4 in [**?**].