CSIT 6000Q- Blockchain and Smart Contracts

Assigment #3
(Due: 11:59pm on Sun Day, Dec. 6, 2023)

November 25, 2023

1 Question 1

What is an Ethereum smart contract?

An Ethereum smart contract is a small program that runs on the Ethereum blockchain.

2 Question 2

What makes an Ethereum smart contract so special compared to other programs?

Once an Ethereum smart contract is deployed to the blockchain, it cannot:

- be stopped
- be hacked (as long as the code of the contract is correct)
- be modified (the code is immutable, however the data is)

3 Question 3

Can a smart contract interact with other smart contracts?

Yes.

4 Question 4

Can a Solidity smart contract call an API on the web?

No, it can only execute its own code and interact with other smart contracts on the blockchain.

5 Question 5

Can a Solidity smart contract store a lot of data?

No, storage is very limited on a smart contract. Storing data cost gas, and gas consumption is capped in each Ethereum block. So indirectly, storage is limited.

6 Question 6

Can a smart contract be written in another language than Solidity?

Yes. There are other smart contract languages like Vyper or LLL. But Solidity is the most popular.

7 Question 7

Is Solidity a dynamically or statically typed language? (i.e need to define variable types)

It's a statically typed language, i.e variable types need to be defined, unlike in dynamic languages like Javascript.

8 Question 8

Is Solidity a dynamically or statically typed language? (i.e need to define variable types)

It's a statically typed language, i.e variable types need to be defined, unlike in dynamic languages like Javascript.

9 Question 9

Is Solidity compiled or interpreted?

It's compiled, meaning it means to be first compiled before we can run the code.

10 Question 10

What is the file extension of Solidity files?

.sol

11 Question 11

What is the typical layout of a Solidity smart contract?

```
//pragma statement (required)
pragma solidity ^0.8.20;

//contract declaration (required)
contract A {
  //state variables
  uint a;

  //functions
  function foo() {...}
}
```

12 Question 12

What is the typical layout of a Solidity smart contract?

```
contract A {
  //state variable
  uint a;

  //functions
  function foo() {
    uint b; //local variable
  }
}
```

State variables are persisted on the blockchain after a smart contract finish to execute, whereas local variables live only during the execution of a function.

13 Question 13

What is the problem with the following code

```
contract Storage {
  uint data;
  //Should update the ‘data‘ storage variable above
  function set(uint _data) external {
    address data = _data;
  }
}
```

The set() function redefine the data variable inside its body. This will create a local variable that will shadow the state variable defined above. The assignment address data = _data will assign to this local variable instead of the state variable. To fix the code, remove the address keyword in front of data in the set() function: data = _data;

14 Question 14

What is the problem with the following code?

```
contract Storage {
  uint data;
  //Should update the ‘data‘ storage variable above
  function set(uint data) external {
    //data = data?
  }
}
```

This is similar to the problem we have just before. The data argument of the set() function shadows the data state variable. Because of this, we can't access the data state variable inside the set() function. To solve this problem, we need to rename the argument from data to _data.

15 Question 15

What are the 2 variable visibilities for state variables in Solidity?

- private
- public

16 Question 16

Who can read private and public variables?

- private variables can be read only by functions inside the smart contract
- public variables can be read by anyone

17 Question 17

What is the default visibility of state variables?

Private.

18 Question 18

Are private variables really private?

No. Private is only for the EVM (Ethereum Virtual Machine, the part of Ethereum that execute smart contracts). But the data of smart contracts is put on the Ethereum blockchain, and all data on this blockchain is public. Using a special tool to analyse blockchain data, anyone to read private variables of smart contracts.

19 Question 19

How to deal with private data then?

You either don't put private data on the blockchain, or you put hashes.

20 Question 20

Mention 3 data types that you use often, and explain why?

- uint, for ether / token transfers, and identifying data
- address, for identifying humans and smart contracts
- strings, for naming things

21 Question 21

What are the 2 container types in Solidity?

- mapping
- arrays

22 Question 22

How to declare an array of integer in Solidity?

```
uint[] a;
```

23 Question 23

How to declare a mapping of address to booleans in Solidity?

```
mapping(address => bool) a;
```

24 Question 24

How to declare a mapping of address to mapping of address to booleans (nested mapping)?

```
mapping(address => mapping(address => bool)) a;
```

25 Question 25

How to declare a mapping of address to mapping of address to booleans (nested mapping)?

```
mapping(address => mapping(address => bool)) a;
```

26 Question 26

How to add data to an array declared as a state variable?

```
uint[] a;

function add(uint newEntry) external {
  add.push(a);
}
```

27 Question 27

How to add data to a mapping declared as a state variable?

```
mapping(address => bool) a;

function add(address addr) external {
  a[addr] = true;
}
```

28 Question 28

How to loop through an array?

```
uint[] a;
for(uint i = 0; i < arr.length; i++) {
  //do something with arr[i]
  //reading: uint a = arr[i]
  //writing: arr[i] = a
}
```

29 Question 29

What is the difference between a uint8 and a uint16?

uint8 can store number of up to $2^8 - 1$ (it has 8 bits), whereas uint16 can store numbers of up to $2^16 - 1$.

30 Question 30

What are the 4 function visibilities in Solidity, by increasing permissiveness?

- private
- internal
- external
- public

31 Question 31

How to conditionally throw an error, with an error message?

```
require(a !== b, 'My error message')
```

32 Question 32

What are the 2 artifacts produced by the Solidity compiler when compiling a smart contract?

- The ABI (application binary interface)
- The bytecode

33 Question 33

What is the ABI of a smart contract?

The ABI defines the interface of a smart contract, i.e the set of functions that can be called from outside the smart contract. The ABI only defines the function signatures (function names, argument types and return types) but not their implementation. The ABI also defines the events of the contract. The ABI is used outside the smart contract by Ethereum client libraries like web3 to interact with the smart contract.

34 Question 34

In the following contract, which function will be part of the ABI?

```
contract A {
  function foo() external {...}
  function bar() public {...}
  function baz() internal {...}
}
```

foo() and bar() will be part of the ABI.

35 Question 35

Does the EVM understands Solidity?

No. The EVM only understand bytecode, which must first be produced by Solidity, outside of the blockchain.

36 Question 36

What is the EVM bytecode?

The EVM bytecode is a series of EVM elementary instructions called opcodes. These opcodes define very simple operations like adding 2 numbers (ADD), loading data from memory (mload), etc... There are more than 100 of these opcodes defined in the Ethereum yellow paper. Coding directly with opcodes would be very tedious, so we need higher languages like Solidity to help us reason at a higher level of abstraction.

37 Question 37

What are the 2 APIs used to interact with a smart contract?

eth_sendTransaction (transaction) and eth_call (call). Transactions cost money (gas) and can modify the blockchain. Calls do not cost money, cannot modify the blockchain, but can return a value contrary to transactions.

38 Question 38

What is gas?

Gas is an abstract unit used to pay miners when sending a transaction on the Ethereum network.

39 Question 39

How is gas paid?

Gas is paid in ether using the formula: ether cost = gasPrice * gas, where gas represents the gas cost of the execution of a transaction. gasPrice is in wei / gas, generally express is Gwei. A transaction also specifies a gasLimit parameter, which specify a maximum number of gas that can be paid by a transaction. Without this, a transaction could potentially drain an account of all its Ether.

40 Question 40

What happen if there is not enough gas in a transaction?

The transaction is stopped, and all state changes are reverted.

41 Question 41

Who pays for gas in a transaction?

The sender of the transaction.

42 Question 42

What do you need to deploy a smart contract to the Ethereum network?

- bytecode of smart contract
- an Ethereum address with enough Ether
- A wallet to sign the transaction
- A tool to create the transaction and coordinate the signing process with the wallet

43 Question 43

List 4 famous Ethereum wallets

- Metamask
- ..
- ..
- ..

44 Question 44

List 3 networks where you can deploy a Solidity smart contract

- Mainnet
- Ropsten
- Goerli

45 Question 45

List 3 networks where you can deploy a Solidity smart contract

- Mainnet
- Ropsten
- Goerli

46 Question 46

List 3 networks where you can deploy a Solidity smart contract

- Mainnet
- Ropsten
- Goerli

47 Question 47

How to manage dates in Solidity?

You need to use uint variables.

48 Question 48

How to have the current timestamp in seconds?

You need to use the now keyword.

49 Question 49

What are the 2 ways to define custom data structure in Solidity?

- Struct
- Enum

50 Question 50

When would you use a struct vs an enum?

Struct are for representing complex data structures with different fields. Enum are for creating variant for a single data. Ex: a color can be red, blue, yellow. You can combine both by defining an enum, and a struct that uses this enum in a field;

```
enum FavoriteColor {Blue, Red};
struct User {
  address id;
  string name;
  Color favoriteColor;
}
```

51 Question 51

What are the 2 ways to instantiate a struct?

```
struct User {
  address id;
  string name;
}

//Method 1 (argument order matters)
```

```
User("0xAio90....", "Mike");

//Method 2 (argument order does not matter)
User({name: "Mike", address: "0xAio90...."});
```

52 Question 52

How to instantiate a struct that has an inner mapping?

```
struct User {
  address id;
  string name;
  mapping(address => bool) friends;
}
//let assume the User struct is stored inside a mapping
mapping(address => User) users;

//Inside a function, you would instantiate your struct like this
users["0xAio90..."].id = "0xAio90...";
users["0xAio90..."].name = "Mike";
users["0xAio90..."].friends["0xIopop..."] = true;
users["0xAio90..."].friends["0xjk89I..."] = true;
```

53 Question 53

When would you use an array vs a mapping?

I would use an array if I need to iterate through a collection of data. And I would use a mapping if I need to rapidly lookup a specific value

54 Question 54

How to combine array and mapping to allow both iteration and rapid lookup of a struct?

```
//Let's consider this struct
struct User {
  uint id;
  string name;
}

//First, let's use an array to store all its ids
```

```
uint[] userIds;

//Then, let's use a mapping for rapid lookup
mapping(uint => User) users;

//If we need to rapidly lookup a user, we use the mapping
//And if we need to iterate through users,
//we iterate through the userIds array,
//and for each userIf we can
//lookup the correct user in the mapping
```

55 Question 55

How to define an in-memory array of 3 integers?

```
uint[] memory arr = new uint[](3);
```

56 Question 56

How to add a value to an in-memory array?

```
uint[] memory arr = new uint[](3);

uint[0] = 1;
uint[1] = 2;
uint[2] = 3;
uint[3] = 1; //out-of-bounds error
```

57 Question 57

How to create an in-memory mapping?

You can't. Solidity has only storage mappings.

58 Question 58

What happen if you try to access the key of a mapping that does not exist?

Contrary to arrays, there is no error, Solidity will give you a value, which is the default value of the type.
Ex:

```
mapping(uint => bool) myMap;
```

If you access myMap[10] but nothing exist there, Solidity will produce false.

59 Question 59

What are the 3 mechanisms for code re-use in Solidity?

- Group common codes in functions
- Contract inheritance
- Libraries

60 Question 61

How to make a contract A inherit from a contract B in Solidity?

```
// First import the contract
import B from 'path/to/B.sol';

//Then make your contract inherit from it
contract A is B {

  //Then call the constructor of the B contract
  constructor() B() {}
}
```

61 Question 62

If A inherit from B, and both define the same function foo, which one will be resolved?

//Case 1

```
contract B {
   function foo() external {...}
}
contract A is B {
   function foo() external {...}
}
```

If I call foo() on A, the function A.foo() will be resolved
//Case 2

```
contract B {
   function foo(uint data) external {...}
}
contract A is B {
   function foo() external {...}
}
```

If I call foo(1) on A, the function B.foo() will be resolved, because only B defines foo(uint)

62 Question 63

What are the 4 memory locations of Solidity?

63 Question 62

If A inherit from B, and both define the same function foo, which one will be resolved?

Storage, Memory, Stack and Calldata

64 Question 63

What is the default visibility of state variables?

Private

65 Question 64

What is the difference between address and address payable?

Only address payable can receive money

66 Question 66

Is it necessary to make an address address payable to transfer ERC20 tokens?

No. The payable requirement is only required for native Ether. Ethereum has no knowledge of ERC20 tokens. For Ethereum, this is just a variable in a smart contract, like any other variables.

67 Question 67

Give 3 ways to save gas

- Put less data on-chain
- Use events instead of storage
- Optimal order for variable declaration.

68 Question 68

How would optimally order uint128, bytes32 and another uint128 to save gas?

- uint128
- uint128
- bytes32

69 Question 69

How would optimally order uint128, bytes32 and another uint128 to save gas?

- uint128
- uint128
- bytes32

The EVM stores variable in 32-bytes slot. However Solidity is smart enough to pack into a single slot several variables if they can fit together. For this optimization to work, packed variables have to be defined next to each other. In the above example, the 2 uint128 will be placed in the same 256 bit slots (128 + 128 = 256).

70 Question 70

How to concatenate 2 strings a, b?

Use the abi.encodePacked() function: `string(abi.encodePacked(a, b));`

71 Question 71

How to get the length of a string in solidity?

```
bytes memory byteStr = bytes(a); //a is a string
  bytesStr.length;
```

72 Question 72

How to to create a smart contract from a smart contract?

```
contract A {
  constructor(uint a) {...}
  function foo() external {...}
}

contract B {
  function createA(uint a) external {
    A AInstance = new A(a); //pass constructor argument(s) if any
  }
}
```

73 Question 73

How to to call another smart contract from a smart contract?

```
contract A {
  function foo() view external returns(uint) {...}
}

contract B {
  function callFoo(address addrA) external {
    uint result = A(addrA).foo();
  }
}
```

74 Question 74

How to get the address of a smart contract that was deployed from a smart contract?

Using the address() operator to cast the contract type into an address:

```
address childAddress = address(new Child());
```

75 Question 75

What will be the value of msg.sender if a contract calls another one?

Using the address() operator to cast the contract type into an address:

```
//This is the inner contract
contract A {
  function bar() view external returns(address) {
     //What will be the value of 'msg.sender' here?
  }
}

//This is the outer contract
contract B {
    function foo() external {
        A aInstance = new A();
         aInstance.bar();
    }
}
```

This is the address of the calling contract, i.e B in our example.

76 Question 76

How to transfer ERC20 tokens?

Using the address() operator to cast the contract type into an address:

```
contract ERC20Interface {
  function totalSupply() public view returns (uint);
  function balanceOf(address tokenOwner)
           public view returns (uint balance);
  function allowance(address tokenOwner, address spender)
           public view returns (uint remaining);
  function transfer(address to, uint tokens)
           public returns (bool success);
  function approve(address spender, uint tokens)
```

```
            public returns (bool success);
  function transferFrom(address from, address to, uint tokens)
            public returns (bool success);

  event Transfer(address indexed from,
             address indexed to, uint tokens);
  event Approval(address indexed tokenOwner,
             address indexed spender, uint tokens);
}

contract DecentralizedExchange {
  function transferToken
       (address ERC20Address, address to, uint amount) {
    ERC20Interface(ERC20Address).transfer(to, amount);
  }
}
```

This is the address of the calling contract, i.e B in our example.

77 Question 77

How to declare and emit an event?

Using the address() operator to cast the contract type into an address:

```
contract A {
  //declare event
  Event TokenSent(uint amount, address to);
  function sendToken(uint amount, address to) external {
      ...
      //Emit event
      emit TokenSent(amount, to);
      //careful, old Solidity 0.4 code didnt not require
         the emit keyword, dont be confused
  }
}
```

This is the address of the calling contract, i.e B in our example.

78 Question 78

What is the indexed keyword in event definition?

If an event field is declared with the indexed keyword, it means that external entities can filter only events whose field match a specific value. For example, in the below example, it means it's possible to filter events with a to field equal to a specific value.

```
Event TokenSent(uint amount, address indexed to);
```

79 Question 79

Is it possible for a smart contract to read the events emitted before?

No. Only external entities can queries events.

80 Question 80

Is it possible to delete or modify a past event?

No. Events are immutable.

81 Question 81

How would you implement access control without modifier?

```
contract A {
  address admin;
  constructor() {
     admin = msg.sender;
  }

  function protectedFunction() external {
    require(msg.sender == admin, 'only admin');
    ...
  }
}
```

82 Question 82

How would you implement access control WITH modifier?

```
contract A {
  address admin;
  constructor() {
     admin = msg.sender;
  }
```

```
  function protectedFunction() external onlyAdmin() {
    ...
  }

  modifier onlyAdmin() {
    require(msg.sender == admin, 'only admin');
    _;
  }
}
```

83 Question 83

How to cancel a transaction?

Once a transaction has been sent, nobody can prevent it from being mined by a miner. But you can still send another transaction preventing the first one from working IF its mined before the first one. This second transaction will have the following properties:

- it will have the same nonce (i.e an incrementing integer that is sent in each transaction, specific to each Ethereum address)
- it will have a higher gasPrice than the first transaction
- it will send a tiny amount of Ether to another address

Let's review why we need these. The same nonce means that the first transaction to be mined will prevent the other one from being mined: miners only mine transactions whose nonce is higher than the previous nonce for the address that has signed the transaction.

The higher gasPrice means a higher reward for miner, so if a miner has the choice to mine the second or the first transaction he will choose the second one.

And finally, sending a tiny amount of Ether is just because a transaction needs to do something on the blockchain, so we just do something that is useless but doesn't cost us much. Actually, you could even call a read-only function in any smart contract, in a transaction, and you wouldn't even need to send this tiny amount of Ether. You would still need to cover the gas fee in every case.

84 Question 84

Is it possible to send a transaction without requiring users to pay gas?

Yes. You would ask users to first sign a message on the frontend. Then the message and signature would be sent to a centralized backend (your app, off-chain) that would create a transaction and embed the payload (message + signature) into it. That means that gas fees will

be covered by the wallet of the app, instead of the user wallet. On-chain, a smart contract will verify the validity of the signature and perform on operation on behalf of the user.

85 Question 85

Which Solidity function would you use to verify a signature?

ecrecover().

86 Question 86

What is a library in Solidity?

A library is a piece of code that be re-used by other smart contracts. There are 2 types of libraries:

- deployed
- embedded

Deployed libraries have their own address, and they can be used by several other smart contracts. Embedded libraries don't have their own address and are deployed as part of the code of the smart contract that use them.

87 Question 87

Give an example of how to use a library in a smart contract

```
library Lib {
  function add(uint a, uint b) pure internal returns(uint) {
    return a + b;
  }
}

contract A {
  using Lib for uint;

  function add(uint a, uint b) pure external returns(uint) {
    return a.add(b);
  }
}
```

88 Question 88

When is a library embedded vs deployed?

```
//Embedded (function is internal)
library Lib {
  function add(uint a, uint b) pure internal returns(uint) {
    return a + b;
  }
}

//Deployed (function is public)
library Lib {
  function add(uint a, uint b) pure public returns(uint) {
    return a + b;
  }
}
```

89 Question 89

When is a library embedded vs deployed?

```
//Embedded (function is internal)
library Lib {
  function add(uint a, uint b) pure internal returns(uint) {
    return a + b;
  }
}

//Deployed (function is public)
library Lib {
  function add(uint a, uint b) pure public returns(uint) {
    return a + b;
  }
}
```

90 Question 90

What is a re-entrancy attack?

A re-entrancy attack happen when a contract A calls a contract B which call back the calling function on contract A to perform some malicious effect. Example with a DAO-like attack:

```
contract A {
  //...
  function pay(address payable to, uint amount) external {
    if(amount <= balances[msg.sender]) {
      B(to).badFunction().send(amount);
      balances[msg.sender] -= amount;
  }
}


contract B {
  address
  function badFunction(address payable to) external {
    ContractA(msg.sender).pay();
  }
}
```

91 Question 91

How to prevent against a re-entrancy attack?

- Solution 1: Decrease balances / do other state variable update BEFORE calling the other contract.
- Solution 2: Put in place re-entrancy guard with a variable that knows when a call is the second in the stack
- Solution 3: Limit the gas available to the called contract. If using transfer(), this is done automatically:

92 Question 92

How to produce a hash of multiple values in Solidity?

```
keccak256(abi.encodePacked(a, b, c))
```

93 Question 93

How to generate a random integer in Solidity?

We can leverage the block.timestamp and block.difficulty as a source of randomness, and use the keccak256() hashing function:

```
uint(keccak256(abi.encodePacked(block.timestamp,
, block.difficulty)))
```

Be aware that miners can manipulate block.difficulty and block.timestamp, so this is not 100% secure.

94 Question 94

How to generate a random integer in Solidity?

We can leverage the block.timestamp and block.difficulty as a source of randomness, and use the keccak256() hashing function:

```
uint(keccak256(abi.encodePacked(block.timestamp,
, block.difficulty)))
```

Be aware that miners can manipulate block.difficulty and block.timestamp, so this is not 100% secure.

95 Question 95

How to declare assembly code?

Functional and instructional. Functional uses functions, whereas instructional is a raw series of opcodes. Most people use the functional style.

96 Question 96

How to declare assembly code?

```
function isHuman(address addr) external {
    uint256 codeLength;

    assembly {codeLength := extcodesize(addr)}
    return codeLength == 0 ? true : false;
  }
}
```