CS-482: Introduction to Blockchain and Cryptocurrency

Serial No:

Sessional Exam 2

Total Time: 1 Hour

Total Marks: 50

Monday, 23rd November 2020

Course Instructors

Dr. Ehtesham Zahoor

Signature of Invigilator

Student Name Roll No Section Signature

DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

Instructions:

1. Attempt on question paper. Attempt all of them. Read the question carefully, understand the question, and then attempt it.
2. No additional sheet will be provided for rough work.
3. After asked to commence the exam, please verify that you have six (6) different printed pages including this title page. There is only one question.
4. Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.
5. For all the questions, you need to be as specific as possible and provide the answers in the space provided. Anything written outside the allocated space (box) would not be considered.
6. Please make sure to properly comment your code and write clearly, failure to do so can cost you 10 marks. Answers containing overwritten text and corrections cannot be claimed for rechecking.

The paper is open book and you can use any printed helping material. Keep your threshold clean and do not share anything. You cannot use any electronic device including calculator.

	Q-1	Total
Marks Obtained		
Total Marks	50	50

Question 1 [50Marks]

Note: For all the questions, you need to be as specific as possible and provide the answers in the space provided. Anything written outside the allocated space (box) would not be considered. Please make sure to properly comment your code and write clearly, failure to do so can cost you 10 marks. Answers containing overwritten text and corrections cannot be claimed for rechecking.

For this question you need to write a Smart Contract for the Escrow transactions scenario, somewhat similar to the one we studied as an example of Bitcoin applications. However, in this case study we would use the contract logic instead of MULTISIGs as in the Bitcoin example. Let us consider Bob to be an online merchant willing to use the Ethereum framework for selling items. He intends to offer escrow payments to gain customers' trust. You, a brilliant student (I know… but let's assume for the time being) of the IBC course, can help him by writing a smart contract. Your contract however would be generic and anyone, say Charlie, can use it as well. The name of the contract is *EscrowPayments*[1] and you need to complete the contract below. You should make all state variables and functions public.

I. Whoever deploys the contract becomes its owner, in our case it is Bob (his account address to be precise). Update the contract below for handling this requirement: **[2 marks]**

```
pragma solidity >=0.4.0 <0.7.0;
pragma experimental ABIEncoderV2; //Hint (or distraction): Allows returning arrays from functions

contract EscrowPayments {

/*The space below is reserved for state variables (all made public) and the Constructor.  For each
state variable that you include please provide a comment specifying the part number, such as the one
below. */

//Part I
```

[1] You may find it easier to read the truffle console logs at the last page to get an idea about the overall flow of the contract.

II. Only the owner of the contract can add items to be sold, along with their prices (in ethers). Write a public function named addItem to add a new item. The function takes two arguments, title and the price. However, to store a new item we need its status and buyer as well.[2] The function adds default values for these at the start, i.e. the item is 'Available' and 'buyer' address is zero. You may also need to update the state variables in the space given after part a, make sure to add a comment specifying part II for any variables added for this requirement. **Hint (or distraction):** Structs in Solidity can be declared and instantiated using the syntax below: **[6 marks]**

```
struct Student {
  string name;
  uint rollno;
}
Student memory ibcStudent = Student('untitled', 1234);
```

III. Anyone can view items, write a function named listItems which returns the list of items.
Hint (or distraction): Iterating through items consumes a lot of gas and even the string concatenation is not straight forward in Solidity. **[4 marks]**

IV. Should the listItems function be made a pure function? If so, update the function signature in the previous part. What is the difference between constant, pure and view functions? **[4 marks]**

[2] You may find it easier to read the truffle console logs at the last page to get an idea about the overall flow of the contract.

V. As our contract is for escrow payments, where a trusted third party (TTP) handles the disputed cases. Add a function to add the address of TTP. Only owner can invoke the function and only once. You may also need to update the state variables in the space given after part I, make sure to add a comment specifying part V for any variables added for this requirement. **[2 marks]**

VI. Any user, say Alice, can buy the items by sending ethers. Write a function named buyItem, with item title as the parameter, which can receive payments and can be invoked by anyone. As our contract is for escrow payments, the payments made by the user are added to the contracts balance and can only be redeemed by the owner once the user confirms the eventual off-chain reception of the item. More specifically, following are the requirements for this function: **[8 marks]**

The buyer needs to send at least the same ethers (more is for sure acceptable) as the price of the item, with the transaction. Otherwise, the purchase is unsuccessful.
If the purchase is successful, the ethers sent with the transaction are added to the contract's balance.
The state of the item needs to be changed to Pending.
You need to handle any case where the item is not found (and ether sent are added to the contract's balance).

Hint (or distraction): keccak256(bytes(a)) == keccak256(bytes(b)) compares string a and the string b.

VII. Once an item has been bought by Alice (or any buyer), she eventually receives the item from the Bob through some offline mode of delivery. The buyer can then confirm the purchase of item to be successful (item is the same as expected) or unsuccessful (the item is probably missing, faulty or similar cases). You need to write a function named confirmPurchase, which takes item title and a boolean parameter signifying the success status. The function can only be invoked by the same address which has made the purchase. If the bool parameter passed is true, the status of the item becomes Confirmed. Otherwise, the state of the item becomes Disputed. **[8 marks]**

VIII. If the state of the item becomes Disputed, as result of confirmPurchase function with false as argument, neither owner nor buyer can withdraw the amount. The case is settled outside our chain by all parties including TTP. The TTP can then change the status of the item from Disputed to either Confirmed or Return. You need to write a function named handleDispute, which takes item title and status as the parameters, for this requirement. The function can only be invoked by the TTP and only if the status of the item is Disputed. **[6 marks]**

IX. Once the item has been confirmed as a successful purchase, having status as Confirmed, the owner can withdraw the price of the item from the contract. Similarly, once the item status has changed to Return, as decided by the TTP for the disputed cases, the buyer can withdraw the price of the item from the contract. You need to write a function named receivePayment, which takes item title as the parameter, for this requirement. The function can only be invoked by owner and the buyer of the item. The status of the item then changes to Expired. **[10 marks]**

Truffle console log for the case of refund

```
truffle(ganache)> let instance = await EscrowPayments.deployed()
truffle(ganache)> instance.owner().then(value=>value.toString())
'0x9a8feDaE0f27EF4ffe41EBDcBaAcf1F14b1902Cb' //The owner is accounts[0]
truffle(ganache)> instance.addTTP(accounts[2])
truffle(ganache)> instance.trustedTP().then(value=>value.toString())
'0xe7589C31f30dE9DbC40975b082e2C50d7ba1Ac22' //trustedTP is accounts[2]
truffle(ganache)> instance.addItem('ItemA',10)
truffle(ganache)> instance.addItem('ItemB',20)
truffle(ganache)> instance.addItem('ItemC',30)
truffle(ganache)> instance.listItems().then(value=>value.toString())
'ItemA,10,0x0000000000000000000000000000000000000000,0x41,ItemB,20,0x0000000000000000000000000000000000
000000,0x41,ItemC,30,0x0000000000000000000000000000000000000000,0x41'
//There is zero address for the buyer and 0x41 is 'A' in hex
truffle(ganache)> instance.buyItem('ItemC', {value:web3.utils.toWei('30','ether'), from:accounts[1]})
truffle(ganache)> instance.listItems().then(value=>value.toString())
'ItemA,10,0x0000000000000000000000000000000000000000,0x41,ItemB,20,0x0000000000000000000000000000000000
000000,0x41,ItemC,30,0x8747FE15B3Da47A9F95DE24444AcA2dcB0242385,0x50'
//The address is of accounts[1] and 0x50 is 'P' in hex
truffle(ganache)> web3.eth.getBalance(accounts[1])
'69999233920000000000'
truffle(ganache)> instance.confirmPurchase('ItemC',false, {from:accounts[1]})
truffle(ganache)> instance.listItems().then(value=>value.toString())
'ItemA,10,0x0000000000000000000000000000000000000000,0x41,ItemB,20,0x0000000000000000000000000000000000
000000,0x41,ItemC,30,0x8747FE15B3Da47A9F95DE24444AcA2dcB0242385,0x44'
//0x44 is D
truffle(ganache)> instance.handleDispute('ItemC','0x52',{from:accounts[2]})
truffle(ganache)> instance.listItems().then(value=>value.toString())
'ItemA,10,0x0000000000000000000000000000000000000000,0x41,ItemB,20,0x0000000000000000000000000000000000
000000,0x41,ItemC,30,0x8747FE15B3Da47A9F95DE24444AcA2dcB0242385,0x52'
truffle(ganache)> instance.receivePayment('ItemC', {from:accounts[1]})
'ItemA,10,0x0000000000000000000000000000000000000000,0x41,ItemB,20,0x0000000000000000000000000000000000
000000,0x41,ItemC,30,0x8747FE15B3Da47A9F95DE24444AcA2dcB0242385,0x58'
//0x58 is X
truffle(ganache)> web3.eth.getBalance(accounts[1])
'99997754160000000000'
```