National University of Computer and Emerging Sciences

Islamabad Campus

(CS-4049) Blockchain & Cryptocurrency (BS-CS)

Final Exam Part B

Date: May 28th, 2024

Total Time (Hrs): 1.5

Total Marks: 60

Total Questions: 3

Course Instructor:

Ms. Urooj Ghani

_____ _____ _____

Roll No **Course Section** **Student Signature**

DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

Instructions:

1. This is Part B of your Exam, estimated time to complete this part is 90 minutes.
2. Read the questions carefully, understand the question, and then attempt it on answer sheet.
3. **Any answer provided on question paper will not be considered for marking. Therefore, avoid writing anything on question paper as final answer.**
4. After asked to commence the exam, please verify that you have **three (3)** different printed pages including this title page. There are a total of **three (3)** questions.
5. Calculator sharing is strictly prohibited.
6. Use permanent ink pens only. **Any part done using soft pencil will not be marked and cannot be claimed for rechecking.**
7. For additional 5 marks, please address all parts of a question in a single response. And mention the question and part no for each solution clearly on top of each answer. Write your roll number clearly on both answer sheet and question paper.
8. Use a permanent pen to clearly cross out any rough work.

Attempt all the questions.

Question 1: **(25 Marks)**

Write a smart contract in solidity to manage the alumni network for batch 20 students who are expected to graduate from FAST-NUCES after the Spring 2024 semester. Below are the updated specifications you need to follow:

Contract Name: [2 Mark]

- AlumniNetwork

Variables to Declare: [2 x 4 = 8 Marks]

- **alumniList:** stores the addresses of alumni from batch 20
- **flexStatus:** to track the Flex status of each alumni (true if active, false if removed)
- **graduationDate:** to store the graduation date of each alumni
- **owner:** the contract owner address

Constructor: [2 Mark]

- Write a constructor to initialize the contract owner.

Modifier: [3 Mark]

Implement a modifier named **onlyOwner** to restrict access to certain functions to the contract owner only. Additionally, specify that only the contract owner **(Sir Amir)** can modify certain aspects of the contract.

Functions to Implement: [5 x 2 = 10 Marks]

- Implement a function named **addAlumni** to add alumni addresses and graduation dates to the network.

- Implement a function named **removeFromFlex** to mark alumni as removed from the Flex program after their graduation.

Instructions:

- Ensure that your Solidity code is clear, well-commented, and follows best practices for readability and maintainability.
- Use appropriate naming conventions for variables, functions, and modifiers.

Solution:

```
pragma solidity ^0.8.0;

contract AlumniNetwork {

  // This is the address of the contract owner (Sir Amir)

  address public owner;
```

```
    // This array will store the addresses of alumni from batch 20
    address[] public alumniList;

    // This mapping will track the Flex status of each alumni (true if active, false if removed)
    mapping(address => bool) public flexStatus;

    // This mapping will store the graduation date of each alumni
    mapping(address => string) public graduationDate;

    // Constructor function to set the contract owner when the contract is deployed
    constructor() {
        owner = msg.sender;
    }

    // Modifier to restrict access to the contract owner only
    modifier onlyOwner() {
        require(msg.sender == owner, "Only the contract owner can call this function");
        _;
    }

    // Function to add an alumni to the network
    // Only the contract owner can call this function
    function addAlumni(address _alumniAddress, string memory _graduationDate) public onlyOwner
{
        // Add the alumni address to the alumniList
        alumniList.push(_alumniAddress);
```

```
        // Set the graduation date for this alumni
        graduationDate[_alumniAddress] = _graduationDate;
        // Mark the alumni as active in the Flex program
        flexStatus[_alumniAddress] = true;
    }

    // Function to remove an alumni from the Flex program
    // Only the contract owner can call this function
    function removeFromFlex(address _alumniAddress) public onlyOwner {
        // Check if the alumni is currently active in the Flex program
        require(flexStatus[_alumniAddress] == true, "Alumni is not in the Flex program");
        // Mark the alumni as removed from the Flex program
        flexStatus[_alumniAddress] = false;
    }
}
```

Question 2: **(10 Marks)**

The smart contract, named FriendCounterBatch20, is designed to keep track of the number of friends each student of batch 20 has. It utilizes a mapping called friendCounts, which associates each student's address with an unsigned integer representing their friend count.
The contract provides three main functions:

addFriend: This function allows a student to increment their friend count by one. It takes the address of the student as an argument and increases their friend count in the friendCounts mapping.

removeFriend: This function enables a student to decrement their friend count by one, but it first checks if the student has any friends to remove. It prevents reducing the friend count below zero to ensure accuracy.

getFriendCount: This function allows anyone to query the friend count of a specific student by providing their address. It returns the current friend count associated with that address from the friendCounts mapping.

```
1  | pragma solidity ^0.8.0;
2 |
3 | contract FriendCounterBatch20 {
```

```
 4 |    mapping(address => uint250) friendCounts;
 5 |
 6 |    function addFriend(address _student) public {
 7 |        friendCounts[–student]++;
 8 |    }
 9 |
10 |     function removeFriend(address _student) public payable {
11 |         require(friendCounts[_student] < 0, "Student has no friends to remove");
12 |         friendCounts[_student]--;
13 |     }
14 |
15 |     function getFriendCount(address _student) public view returns (unint256) {
16 |         return friendCounts[_student];
17 |     }
18 | }
```

Identify and correct the errors in the provided smart contract. Create a table on your answer sheet indicating the line number containing the error and the corrected line. Below is the header of the table for your reference.

Line No	Error	Corrected Line

Question 3: **(4 x 5 = 20 Marks)**

a. Sabeen and Ramsha both want to execute a smart contract simultaneously. What approach should they take? **[5 Marks]**

When Sabeen and Ramsha both want to execute a smart contract simultaneously, they should understand that the Ethereum blockchain handles transactions sequentially, not in parallel. Each transaction is added to a block in a specific order. Therefore, if they both submit transactions at the same time, one transaction will be processed first, and the other will be processed afterward. They do not need to take any special approach; they just need to submit their transactions and the network will handle the order.

b. Taha has deployed a smart contract on the Ethereum blockchain and now wishes to delete it. How can he achieve this, and what happens after the deletion? **[5 Marks]**

Taha can delete a smart contract on the Ethereum blockchain by calling the selfdestruct function within the contract, provided it was implemented by the contract author. This function removes the contract code and its state from the blockchain and returns any remaining Ether in the contract to a specified address. After the deletion, the contract's address will no longer hold any code, and any interactions with this address will be invalid.

c. Huzaifa, Umais and Moonis are working on a project together. They need to ensure compatibility with different versions of Ethereum. To manage this, they decide to follow Ethereum's versioning model. How is Ethereum's versioning model structured? **[5 Marks]**

Huzaifa, Umais, and Moonis should follow Ethereum's versioning model, which is structured as MAJOR.MINOR.PATCH. This model helps manage compatibility by clearly indicating the type of changes made in each version:

MAJOR: Incompatible API changes.

MINOR: Backward-compatible functionality added.

PATCH: Backward-compatible bug fixes.

d. How does censorship resistance help Mahad when using a decentralized application (DApp)? **[5 Marks]**

Censorship resistance helps Mahad when using a decentralized application (DApp) by ensuring that as long as he has access to an Ethereum node, he can interact with the DApp without interference from any centralized authority. This guarantees that no single entity can prevent him from accessing or using the DApp, maintaining the decentralized and open nature of the application.

Best of Luck 😊