IBU International Burch University                                                                 STIRLING EDUCATION

Date: 21 / 11 / 2024        Duration: 90 minutes

Student name: ........................................................

# Blockchain Fundamentals - Midterm Exam 1 (Practical B)

## Task 1 (10%)

Construct a **Merkle tree** from the following input *data points*:

- **vitalik**
- **wei**
- **joseph**

Use the **SHA-256** algorithm in the process, and briefly describe the steps you took.

## Task 2 (30%)

In this task, you will need to code the basic logic of the card game **Uno**.

In case you have not played the game before, here is a breakdown of the basic rules (that you will need for the task). Uno is a turn-based card game in which each player is dealt 7 cards. The cards are colored in one of the **four** colors: **red**, **green**, **blue** and **yellow**, and have numbers **from 0 to 9 (inclusive)** on them. On their turn, a player has to **put down (play) one card** which is either of **the same color** as *the last played card* (but has a *different number*), or has **the same number on it** as the *last played card* (but a *different color*). For example, if Player 1 plays a **red 9**, Player 2 has to either:

- play **any other red card**
- play a **9 of a different color** (**green**, **blue** or **yellow**).

If the player has *no matching cards*, they have to *draw a new card* from the deck. The game ends when one player has *no cards left*.

This is the main gist of how Uno is played. There are certain other rules and special cards, but they are **not important nor relevant** for your task. Your task will be to simulate a basic *addition of players* to the game, *starting / ending* of the game and *card playing logic*. You **do not need** to implement decks, card drawing, player "hands", nor any special cards. **Make sure** to properly *handle all requirements* of certain functions.

Using **Solidity**, you need to do the following:

1. Create an **enum** Color to keep track of card colors: *Red*, *Green*, Blue, *Yellow*
2. Create a **struct** Card which will represent a played card.
   a. A card is defined by a *number* (integer) and a *color* (enum).
3. Create a smart contract called **Uno**, which should keep track of the following information:
   a. an array of *player addresses*

   b. the *last played card*
   c. whether the game has *started or not*
   d. *which player's turn* it is, represented by an *integer* (0 → player 1, 1 → player 2, ...)
   e. the *owner* of the contract
4. Make sure that the following is done when the contract is *first deployed*:
   a. Set the *contract owner*.
   b. "Hard-code" the *last played card* as a **red 1**.
      i. In the actual game, you would take the top card from the deck as your starting card, but this is for simplicity's sake.
5. Create a **custom modifier** that will *prevent non-owners* from calling certain functions.
6. Create a function **addPlayer(address _player)**
   a. The function should take in an address of a new player to add to the game.
   b. Player addresses should be stored in *an array* mentioned in point 3.
   c. A new player *cannot be added* if the game *has already started*.
   d. Only the *original owner* should be able to call this function.
7. Create a function **startGame()**
   a. The function should *mark the game as started*.
   b. It *should not be possible* to start an *already started game*.
   c. The game can only start if there are *2 or more players*. If there are *no players or only 1 player*, the game *cannot* start.
   d. Only the *original owner* should be able to call this function.
8. Create a function **endGame()**
   a. This function should *mark the game as done*.
   b. It *should not be possible* to stop a *game that has not started yet*.
   c. Only the *original owner* should be able to call this function.
9. Create a function **playCard(Card memory _card)**
   a. This function should take in a *Card struct* which represents a card that *a player is playing* on their turn.
      i. **Note**: when testing this function in Remix IDE's "Deploy" tab, you *have to* use this format **[1, 0]** as input. The first number will be the card number, and the second number will be the *ordinal number* of the color from your enum (if your order is red, green, blue, yellow; 0 → red, 1 → green, ...). Hence, [1, 0] would be a **red 1**, [9, 1] is a **green 9**, etc.
   b. A player *cannot* play a card if the game *has not started*.
   c. **Ensure** that a played card is *valid*: the player's card has to either *have the same number* or the same color as the ***last played card*** (by the *previous player*).
   d. **Ensure** that card numbers are valid (from 0 to 9 inclusive).
   e. **Ensure** that a player cannot *play outside of their turn*. Players should take turns based on *the order in which they were added to the game* by the owner.

i. Players "play in turn" by calling the playCard() function *from their account*, e.g. Player A calls playCard(), then Player B calls playCard(), etc.
ii. If players A, B and C are added in the order *[B, A, C]*, they *have to* play in that order; player C cannot play before player A.
iii. **Note:** think about what happens when the last player is done; the "cycle" has to *restart from the first player*.

10. **Deploy** this contract to the **Sepolia testnet**, and **verify** it on **Etherscan**.

## Task 3 (20%)

You are given the following smart contract. It is a simple *magazine subscription service*, which has a function for listing all your subscriptions, subscribing to new magazines and unsubscribing. Each magazine *has an ID*, and upon unsubscribing from a magazine, an event is emitted. The constructor also contains a list of a few default subscriptions to get you started.

Your task is to create a **simple UI** for this smart contract. You can use **any frontend framework** for the application, and choose between either **Web3.js** or **Ethers.js** as the blockchain library. You **do not have to style** the application; it can be pure basic HTML.

You should do the following:

1. **Deploy** this contract to the **Sepolia testnet**, and **verify** it.
2. Create a simple **login via MetaMask**.
   a. Initially, all application data *should be hidden* except a *"Login with MetaMask"* button.
   b. Once a user logs in, the login button should disappear.
3. Upon logging in, the user should see a **list of their subscriptions**.
   a. Each magazine name should be displayed, followed by a button *"Unsubscribe"*.
4. Clicking on the "Unsubscribe" button next to the magazine should **call the function** on the contract to *unsubscribe from that particular magazine*.
5. Once you unsubscribe from a magazine, the contract will emit an **event**; your application should **listen** for that event and upon receiving it, **refresh** the *list of user's subscriptions*.

**Good luck.**