Introduction to Blockchains

CS 426 Assignment 3

Design and Development of Smart Contract Auction for Fundraising for a Social Cause

Assignment Details

- **Assignment Given On:** 23rd March 2025
- **Due Date:** 27th March 2025, on Google Classroom (10% penalty per 24-hour period after the due date).

Submission Format:

- Create a directory named `<Your Roll Number>`.
- Copy all output program files (source code) and relevant screenshots into this folder, all again named with respective roll number.
- Include a Makefile or script file for compilation and execution.
- Zip the directory and submit it on Google Classroom.

Additional Instructions:

- **DO NOT USE CHATGPT OR COPY FROM ANYWHERE ON THE INTERNET OR COPY FROM ANOTHER STUDENT.**
  100% penalty if the submitted source code is found to be copied.

Learning objectives:

- To design, develop and test a smart contract for a problem using Solidity language and Remix IDE.
- To apply incremental development methods and best practices discussed in the course.

Problem Statement:

Consider the problem of Chinese auctions or penny socials. We will refer to it as a simple "Auction". It is a conventional approach used for fund raising for a cause. The organizers collect items to be auctioned off for raising funds. Before the auction, the items for auctions are received and arranged each with a bowl to place the bid. A chairperson is a special person among the organizers. She/he heads the effort and is the only person who can determine the winner by random drawing at the end of the auction. A set of bidders buy sheets of tickets with their money. The bidder's sheet has a stub that identifies the bidder's number, and tokens bought.

The bidders examine the items to bid, place the one or more tickets in the bowl in front of the items they desire to bid for until all the tickets are used. After the auction period ends the chairperson collects the bowls, randomly selects a ticket from each item's bowl to determine the winning bidder for that item. The item is transferred to the winning bidder. Total money collected is the fund raised by the penny social auction.

Assumptions:

The description given above is for a general penny social auction. For the sake of our project implementation we will introduce some simplifying assumptions. Here they are:

1. Fixed number of bidders, initialized to 4. All 4 need to self-register. Funds transfer from bidder is automatically done and is not in the scope of this application.
2. Fixed number of items to be auctioned off, initialized to 3.
3. Items auctioned are indexed from 0..N-1 where N is the number of items for auction. N is 2.
4. Each bidder buys just 1 sheet of tickets or tokens; each sheet has only 5 tokens.
5. Assume simple numbers for the serial numbers for the sheet of tickets: 0,1,2,3. Here we show the tokens of bidder 0 and 1, for n people and so on.

0	0	0	0	0
1	1	1	1	1
.	.	.	.	.
n-1	n-1	n-1	n-1	n-1

The Design:

Let's design the smart contract for this. Visualize the situation using the screen shot given below.

These are the pictures of items, we need to define "Items" in the smart contract. We will also need to define Persons or bidders who will bid on the Items. We will also need some supporting data variables. The functions are similar to the Ballot of our lessons. Constructor that initializes the owner, register that allows (decentralized person) to register online to get the tokens and start bidding, bid function that lets a person bid, and finally revealWinner, to randomly choose the winner for the item auctioned. Here is our design. Always remember to design first. The Auction smart contract has Item and Person structs and other data items such as array of Items, array of Persons, array of winners, mappings, and beneficiary address.

Implementation:

You will write two versions of the implementation templates: version 1: without modifier and version 2 with modifier. We have provided the templates for both versions.

1. Please understand the problems before you proceed.
2. Copy the template into your Remix IDE. Complete the code. You will need to fill in the code at *ONLY* at the locations indicated. Test it to see if it is operational.
3. Implement version of as Auction.sol. Then update this base version for Auction.sol for Part 2 requirements.

Version 1:

Auction.sol: Download the Auction.sol file from shared file. There are 6 tasks where you will fill in the code. You need to review the code given and understand it fully before you start adding code. This partial code is available in the Auction.sol.

Testing on Remix:

Test the completed code by compiling and running it on Remix JavaScript VM.

(i) Account 0 provided by JavaScript VM is the Auction beneficiary and will not bid.

(ii) Navigate to each of the other accounts, and "register." You will register four bidders using the register function.

(iii) The next step is for the bidders to each bid; for test purposes, you can execute the "bid" function with {{0,1}{1,1}{2,1} } for each of the four accounts.

(iv) Next, execute the "revealWinner" function to determine the winner for each item randomly, and

(v) the getter of the "winners" data can be executed with {0, 1 and 2} as a parameter in sequence to reveal the winners of the draw respectively. You can do a lot of more testing and exploration with the buttons provided by the Remix web interface.

Version 2:

For version 2, we need to add a modifier so that only the owner can invoke the function "revealWinner".

```
/*Part 2 Task 1. Create a modifier named "onlyOwner" to ensure that only the owner is
allowed to reveal winners.
Hint: Use require to validate if "msg.sender" is equal to the "beneficiary". */
```

This involves defining a modifier and using the modifier in the header of the "revealWinner" function.

Add these updates, compile, and test with Remix IDE. Make sure that the operations of "revealWinner" from any other account than the Owner (account{0}) results in "revert". Save the completed solution and submit.

Version 1: 6 tasks * 5 Points = 30 points

Version 2: 1 task * 10 Points = 10 points