1 Storage

Problem 1. Based on the contract below, give the locations in storage where the EVM would put
\(a, b[1], c.value, d[25], f[32], g[1][2], h[1][2]\).

Hint: You can think of Solidity storage as an array of \(2^{256}\) slots. You can assume that this
contract’s storage starts at slot 0. Your answer can be in terms of the hash function used.

```solidity
contract StorageTest {
    uint256 a;
    uint256[2] b;
    
    struct Entry {
        uint256 id;
        uint256 value;
    }
    Entry c;
    Entry[] d;
    
    mapping(uint256 => uint256) e;
    mapping(uint256 => uint256) f;
    
    mapping(uint256 => uint256[]) g;
    mapping(uint256 => uint256[]) h;
}
```

Figure 1: Example Solidity Contract

Problem 2. Logically, a mapping in Ethereum is defined over every possible 256-bit key, mapping
each such key to zero. Sketch Solidity psuedocode and explain changes needed to add the ability
to iterate over all key-value pairs in a mapping where the value has been explicitly assigned. Also,
sketch psuedocode or explain how this change would affect inserting and deleting into the mapping.

2 Pitfalls

Problem 3. Figure 2 shows a lottery contract found on the web. Explain how you can determine
\texttt{secretNumber}.

Problem 4. So you know the contract’s \texttt{secretNumber} from the previous exercise. Can you clean
out the contract by calling \texttt{play()} with that number? What will happen if you do?

Hint: Notice that \texttt{secretNumber} is stored at slot 0.
Problem 5. To reduce the threat of a stack overflow attack, Solidity recently imposed a rule that a nested call has access to only $\frac{64}{64}$ of its parents’ gas. Consider the ‘King of the Ether’ contract studied in class. Suppose we want to do a stack overflow attack, calling the contract’s bid() function from a stack depth of 1023, so that the nested call intended to send funds to the previous King will silently fail at depth 1025 (recall that Ethereum has a max stack depth of 1024). We will need 100 gas to complete our call to bid(). How much gas do we need to start with to execute this attack?
pragma solidity ^0.4.23;

// EtherGame

// Guess the number secretly stored in the blockchain and win the whole contract balance!
// A new number is randomly chosen after each try.

// To play, call the play() method with the guessed number (1–10). Bet price:
// 0.2 ether

contract EtherGame {
    uint256 private secretNumber;
    uint256 public lastPlayed;
    uint256 public betPrice = 0.001 ether;
    address public ownerAddr;
    struct Game {
        address player;
        uint256 number;
    }
    Game[] public gamesPlayed;

    constructor () public {
        ownerAddr = msg.sender;
        shuffle();
    }

    function shuffle () internal {
        // randomly set secretNumber with a value between 1 and 10
        secretNumber = uint8(sha3(now, block.blockhash(block.number−1))) % 10 + 1;
    }

    function play(uint256 number) payable public {
        require (msg.value >= betPrice && number <= 10);
        Game game;
        game.player = msg.sender;
        game.number = number;
        gamesPlayed.push(game);
        if (number == secretNumber) {
            // win!
            msg.sender.transfer (this.balance);
        }
        shuffle();
        lastPlayed = now;
    }

    function kill () public {
        if (msg.sender == ownerAddr && now > lastPlayed + 6 hours) {
            suicide (msg.sender);
        }
    }

    function () public payable { }
}

Figure 2: Lottery Honeypot