CODE 1

```
#!/usr/bin/env python
# coding: utf-8

import heapq

# Node structure for Huffman Tree
class HuffmanNode:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    # Defining less than operator for priority queue comparison
    def __lt__(self, other):
        return self.freq < other.freq

# Function to generate Huffman codes
def generate_codes(root, current_code, codes):
    if root is None:
        return
    if root.char is not None:
        codes[root.char] = current_code
    generate_codes(root.left, current_code + "0", codes)
    generate_codes(root.right, current_code + "1", codes)

# Function to build Huffman Tree
def build_huffman_tree(frequency):
    heap = []
    # Insert all characters with their frequencies into the heap
    for char, freq in frequency.items():
        heapq.heappush(heap, HuffmanNode(char, freq))

    # Merge nodes until we have one tree
    while len(heap) > 1:
        node1 = heapq.heappop(heap)
        node2 = heapq.heappop(heap)

        # Create a new internal node with the combined frequency
        merged = HuffmanNode(None, node1.freq + node2.freq)
        merged.left = node1
        merged.right = node2
```

```
        heapq.heappush(heap, merged)

    # The root of the Huffman Tree
    return heapq.heappop(heap)

# Function to calculate frequency of characters
def calculate_frequency(data):
    frequency = {}

    for char in data:
        if char not in frequency:
            frequency[char] = 0
        frequency[char] += 1

    return frequency

# Huffman Encoding process
def huffman_encoding(data):
    frequency = calculate_frequency(data)
    huffman_tree_root = build_huffman_tree(frequency)
    codes = {}
    generate_codes(huffman_tree_root, "", codes)

    # Encode the input data
    encoded_data = "".join([codes[char] for char in data])

    return encoded_data, huffman_tree_root

# Huffman Decoding process
def huffman_decoding(encoded_data, huffman_tree_root):
    decoded_data = ""
    current_node = huffman_tree_root

    for bit in encoded_data:
        if bit == '0':
            current_node = current_node.left
        else:
            current_node = current_node.right

        if current_node.left is None and current_node.right is None:
            decoded_data += current_node.char
            current_node = huffman_tree_root
```

```
    return decoded_data

# Driver code
if __name__ == "__main__":
    data = input("Enter the string to be encoded using Huffman Encoding: ")

    encoded_data, huffman_tree_root = huffman_encoding(data)
    print(f"\nEncoded Data: {encoded_data}")

    decoded_data = huffman_decoding(encoded_data, huffman_tree_root)
    print(f"Decoded Data: {decoded_data}")
```

CODE 2

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Bank {
    // State variable to store the balance of the customer
    mapping(address => uint256) private balances;

    // Event to log deposits
    event Deposit(address indexed account, uint256 amount);

    // Event to log withdrawals
    event Withdrawal(address indexed account, uint256 amount);

    // Function to deposit money into the bank account
    function deposit() public payable {
        require(msg.value > 0, "Deposit amount must be greater than zero");

        // Increase the balance of the sender
        balances[msg.sender] += msg.value;

        // Emit the deposit event
        emit Deposit(msg.sender, msg.value);
    }

    // Function to withdraw money from the bank account
    function withdraw(uint256 amount) public {
        require(amount > 0, "Withdrawal amount must be greater than zero");
        require(balances[msg.sender] >= amount, "Insufficient balance");
```

```
        // Decrease the balance of the sender
        balances[msg.sender] -= amount;

        // Transfer the amount to the sender
        payable(msg.sender).transfer(amount);

        // Emit the withdrawal event
        emit Withdrawal(msg.sender, amount);
    }

    // Function to show the balance of the sender's account
    function showBalance() public view returns (uint256) {
        return balances[msg.sender];
    }
}
```