BlockBloom Project Assignment 1

Manas Todi
Roll Number: 230627

January 8, 2025

Answers

1. Factorial Implementation

Iterative Method: Sepolia ETH used - 0.00384467

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

contract Factorial {
    function fact1(uint x) public pure returns (uint ans) {
        ans = 1;
        for (uint i = 1; i <= x; i++) {
            ans *= i;
        }
    }
}
```

Recursive Method: Sepolia ETH used - 0.003705

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

contract Factorial {
    function factRec(uint x) public pure returns (uint) {
        if (x == 0 || x == 1) {
            return 1;
        }
        return x * factRec(x - 1);
    }
}
```

Theory: Recursive methods generally use more gas due to stack space usage, as each function call creates a new stack frame. Iterative methods update a single variable, consuming less memory and resulting in lower gas fees.

2. Owner Modifier Implementation

Sepolia ETH used - 0.00880713

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

contract Factorial {
    address public own;
    constructor() {
        own = msg.sender;
    }

    modifier Owner() {
        require(msg.sender == own, "Not contract Owner");
        _;
    }

    function updateOwner(address newOwner) public Owner {
        require(newOwner != address(0), "Invalid address for new owner");
        own = newOwner;
    }

    function factorial(uint x) public pure returns (uint result) {
        result = 1;
        for (uint i = 1; i <= x; i++) {
            result *= i;
        }
    }

    function factRec(uint x) public view Owner returns (uint) {
        if (x == 0 || x == 1) {
            return 1;
        }
        return x * factRec(x - 1);
    }
}
```

Modifiers:

- **Owner Modifier:** Enforces access control, ensuring only the contract owner can call specific functions.
- **Visibility Modifiers:** Control access levels (`public`, `private`, `internal`, `external`).
- **Mutability Modifiers:** Define state interactions (`pure`, `view`, `payable`).

3. Error Handling in Solidity

- **Require:** Validates conditions before function execution. If the condition fails, the transaction is reverted and unused gas is refunded.
- **Assert:** Ensures critical conditions inside the contract are true. Failure indicates a bug and consumes all gas.

- **Revert:** Manually stops execution when conditions are unmet, undoing changes and refunding unused gas.

4. Selfdestruct Function

The `selfdestruct` function in Solidity removes a contract's code and storage from the blockchain and transfers any remaining Ether to a specified address. While it helps clean up resources, improper use can lead to risks like unauthorized execution. Secure implementation with access controls is crucial to prevent misuse.

5. Output Behavior in Remix IDE

In Remix IDE, output behavior depends on whether a function interacts with the blockchain state:

- **State-Changing Functions:** Outputs appear after the transaction is mined (e.g., `transferEther()`).
- **View Functions:** Immediate results without mining, as they do not modify the blockchain (e.g., `display()`).