F BUST Code Example - Factorial and Fibonacci Series

Factorial Program

```
// Simple program to calculate the factorial of a number using recursion in F BUST

// Defining the factorial function
function factorial(n) {
    // Base case: factorial of 0 or 1 is 1
    if (n == 0 || n == 1) {
        return 1;
    }
    // Recursive case: factorial of n is n * factorial of (n-1)
    return n * factorial(n - 1);
}

// Main function to demonstrate factorial calculation
function main() {
    // Variable to store the number whose factorial is to be calculated
    let num = 5;

    // Calling the factorial function and storing the result
    let result = factorial(num);

    // Output the result
    print("The factorial of " + num + " is: " + result);
}

// Execute the main function
main();
```

Explanation

1. Functions and Recursion:
   - factorial(n): This function calculates the factorial of a given number using recursion. The base case is when n == 0 or n == 1, where the factorial is 1. Otherwise, it calls itself

with the value n - 1 and multiplies it with n to compute the factorial.

2. Main Function:
   - The main() function initializes a variable num to 5 and calls the factorial() function with that number. The result is stored in result and printed to the console.

3. Recursion and Output:
   - Recursion in F BUST is simple and direct, and the factorial function calls itself until it reaches the base case, showcasing both algorithmic logic and F BUST’s efficiency.

4. Printing:
   - The print() function in F BUST outputs the result to the console, demonstrating the language's ease of output display.

Fibonacci Program with Error Handling

```
// Fibonacci function with error handling
function fibonacci(n) {
    // Check for invalid inputs
    if (n < 0) {
        print("Error: Input must be a non-negative integer.");
        return;
    }

    // Base cases for Fibonacci series
    if (n == 0) return 0;
    if (n == 1) return 1;

    // Recursion to find Fibonacci of n
    let a = 0, b = 1, c;
    for (let i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}

// Main function to display Fibonacci series
```

```
function main() {
    let num = 10;
    let result = fibonacci(num);

    // Displaying the result of Fibonacci calculation
    print("The " + num + "th Fibonacci number is: " + result);
}

// Execute the main function
main();
```

Explanation

1. Error Handling:
   - The function checks if the input n is negative. If it is, the program prints an error message and returns without attempting further calculations. This demonstrates the importance of input validation in professional software development.

2. Fibonacci Calculation:
   - While recursion elegantly models the mathematical definition of Fibonacci numbers, it becomes inefficient for large values of n due to stack overflows. Alternatively, this iterative algorithm circumvents such limitations by dynamically accumulating successive terms in a loop.

3. Efficiency:
   - Rather than recursively invoking itself, the approach accumulates the running sums of the previous two numbers on each iteration. This allows it to compute any Fibonacci number with linear time complexity, solving the problem without recursion's exponential overhead. For parameters beyond the reach of recursion, the iterative nature ensures responsive performance no matter the scale of n.

4. Display Results:
   - The Fibonacci number is printed to the console, just like in the previous program, ensuring clear and readable output.