## 1. Continuous Mean Value Calculation
## Approach - 2

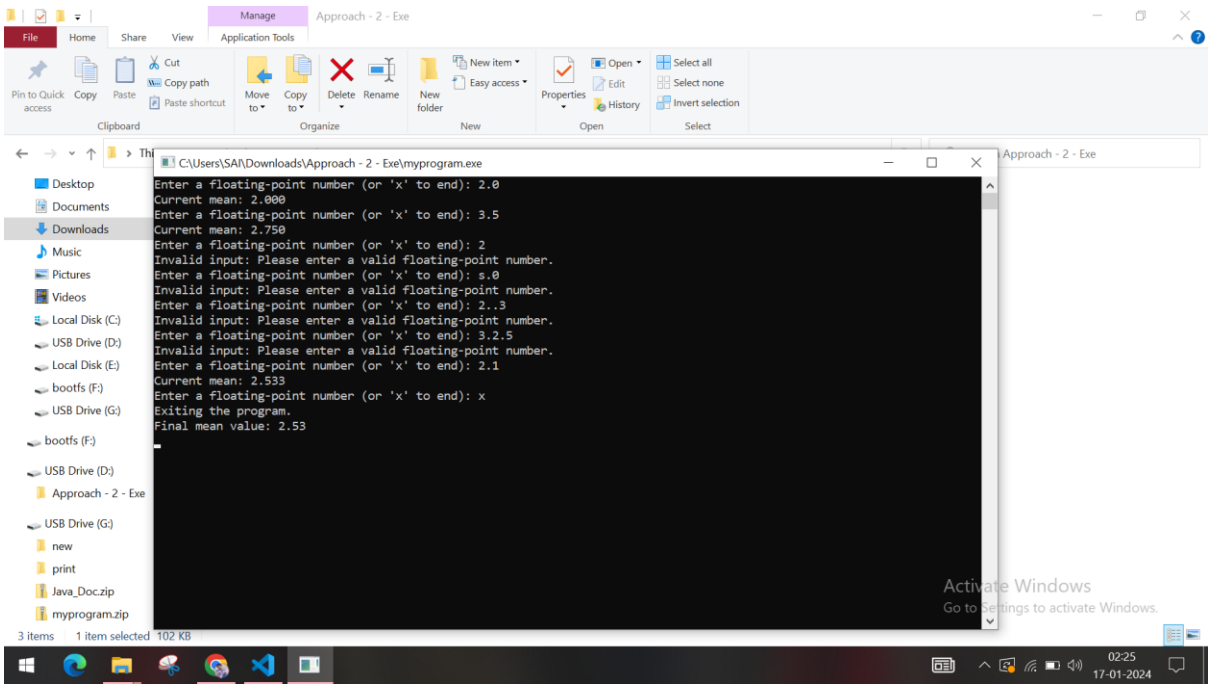
**Explanation**

This C++ program is designed to interactively calculate the mean (average) of a series of floating-point numbers entered by the user. The program consists of a main function and a supporting function, `isValidFloatingPoint`, for input validation.

In the `isValidFloatingPoint` function, a string representing user input is processed using an `std::isstringstream`. The function checks whether the input can be successfully converted to a double and ensures that there are no remaining characters after the conversion. Additionally, it verifies if the input contains a decimal point, returning true if these conditions are met, indicating a valid floating-point number, and false otherwise.

The main function initializes variables for the total sum (`total_sum`) and the count of entered numbers (`count`). It enters a continuous input loop, prompting the user to enter floating-point numbers until the user inputs 'x' to terminate the process. Within the loop, user input is validated using the `isValidFloatingPoint` function. If the input is deemed invalid, an error message is displayed, and the loop continues to the next iteration. Valid input is converted to a double using `std::stod`, and the program updates the sum and count variables. The current mean is then calculated and displayed, with precision control using `std::fixed` and `std::setprecision`.

Exception handling is implemented to catch and handle cases where the conversion of input to a double fails due to an invalid argument. The final mean value is displayed after the user terminates input, and the program introduces a deliberate delay of 5 seconds using `std::this_thread::sleep_for` before exiting. This allows the user a brief moment to review the final result before the program concludes. The program encapsulates input validation, exception handling, and interactive calculation of the mean, providing a straightforward utility for users to analyze a series of floating-point numbers.



**Steps**

1. **Input Validation Function (isValidFloatingPoint)**
   The program begins with a function that validates whether a given string represents a valid floating-point number. This is achieved by using an std::istringstream to attempt conversion to a double and checking for any remaining characters. The presence of a decimal point is also verified. This function is crucial for ensuring that user input is valid before attempting further processing.

2. **Main Function Initialization**
   The main function initializes variables total_sum and count to maintain the cumulative sum of entered numbers and the count of entered values, respectively. These variables are used to calculate the mean.

3. **Continuous Input Loop**
   The program enters a continuous loop where the user is prompted to input floating-point numbers. The loop continues until the user enters 'x' to signify the end of input.

4. **User Input and Termination Check**
   Within the loop, the program uses std::getline to obtain a string input from the user. If the user enters 'x', the program displays an exit message and breaks out of the loop, concluding the input phase.

5. **Input Validation and Error Handling**
   User input is then validated using the isValidFloatingPoint function. If the input is invalid, an error message is displayed, and the program skips to the next iteration of the loop.

6. **Conversion and Update**
   Valid input is converted to a double using std::stod, and the program updates the cumulative sum and count variables. The current mean is calculated and displayed with a precision of three decimal places.

7. **Exception Handling**
   The program includes exception handling to catch cases where the conversion of input to a double fails due to an invalid argument. In such cases, an error message is displayed, allowing the program to gracefully handle exceptional input scenarios.

8. **Final Mean Calculation, Display and Exit**
   After the user terminates input, the program calculates the final mean value based on the accumulated sum and count. This value is displayed with a precision of two decimal places. And the program introduces a 5-second delay using std::this_thread::sleep_for before exiting.

**Code**

```
#include <iostream>
#include <string>
#include <sstream>
#include <iomanip>
#include <limits>
#include <chrono>
#include <thread>

bool isValidFloatingPoint(const std::string& input) {
    std::istringstream iss(input);
    double test;
    char leftover;

    // Attempt to read a double followed by any remaining characters
    if (iss >> test && !(iss >> leftover)) {
        // Check if the input contains a decimal point
        return (input.find('.') != std::string::npos);
    }

    return false;
}

int main() {
    // Variable declaration
    double total_sum = 0.0;
    int count = 0;

    // Continuous input loop
    while (true) {
        // Get a new value from the user
        std::string input_value;
        std::cout << "Enter a floating-point number (or 'x' to end): ";
        std::getline(std::cin, input_value);

        // Check if the user wants to end the input
        if (input_value == "x") {
            std::cout << "Exiting the program." << std::endl;
            break;
        }

        // Validate the input
        if (!isValidFloatingPoint(input_value)) {
            std::cerr << "Invalid input: Please enter a valid floating-point
number." << std::endl;
            continue;
        }
```

```
        try {
            // Convert input to a double
            double new_value = std::stod(input_value);

            // Update variables
            total_sum += new_value;
            count++;

            // Calculate and display the current mean
            double current_mean = total_sum / count;
            std::cout << "Current mean: " << std::fixed <<
std::setprecision(3) << current_mean << std::endl;

        } catch (const std::invalid_argument& e) {
            std::cerr << "Invalid input: " << e.what() << std::endl;
        }
    }

    // Display the final mean value
    double final_mean = (count > 0) ? total_sum / count : 0.0;
    std::cout << "Final mean value: " << std::fixed << std::setprecision(2) <<
final_mean << std::endl;

    // Introduce a delay before exiting (e.g., 5 seconds)
    std::this_thread::sleep_for(std::chrono::seconds(5));

    return 0;
}
```