Problem Description

Servo motors operate on mechanical timescales (milliseconds for movement), while previous code, particularly function calls and control loops, operates at much faster computational timescales (nanoseconds or microseconds). This difference can lead to situations where:

1. **Servo Lag:** If the program tries to execute other tasks or functions while the servo is in motion, it might seem like the servo is not completing its tasks.
2. **Timing Mismatch:** The servo's physical movement might not keep up with computationally intensive tasks or rapid iterations in the loop.

How to Handle Servo Timing in Your Code

1. **Introduce a Non-Blocking Approach (Using Millis):**
   - Avoid `delay()` when sweeping the servo. Use a timer-based approach with `millis()` to allow other parts of the loop to execute while the servo moves.
2. **Use Task Scheduling:**
   - Implement a state machine or task scheduler to manage the servo's movement alongside other tasks.
3. **Prioritize Tasks:**
   - Ensure servo movement has sufficient time to complete before running other high-priority functions.

Key Improvements:

1. **Non-Blocking Movement:**
   - The `moveServo()` function uses the `millis()` timer instead of `delay()`. This allows other tasks to execute while the servo moves.
2. **Dynamic Speed and Angle:**
   - You can adjust the servo speed and target angle on the fly using the Serial Monitor.
3. **Real-Time Responsiveness:**
   - The servo moves in small increments toward the target angle without halting the loop.
4. **Input Validation:**
   - The input parser ensures only valid numeric values are accepted, preventing invalid inputs like `10X` or `11@`.

Modified Code Using Non-Blocking Approach

Here’s your updated code to handle servo movement without blocking the loop:

```
#include <Servo.h>

// Create a servo object to control a servo
Servo myservo;

// Define the pin to which the servo is attached
const int servoPin = 9;

// Servo parameters
int currentAngle = 0;          // Current angle of the servo
int targetAngle = 0;           // Target angle for the servo
unsigned long lastMoveTime = 0; // Time of the last servo movement
const int stepDelay = 20;      // Delay between steps (ms)
bool once = true;

// Define the default speed (delay time between steps in milliseconds)
int servoSpeed = 15;           // Servo speed: lower value = faster movement

void setup() {
 // Attach the servo on the specified pin to the servo object
 myservo.attach(servoPin);

 // Move the servo to the initial position (0 degrees)
 myservo.write(currentAngle);

 // Initialize Serial communication for debugging and speed adjustment
 Serial.begin(9600);
 Serial.println("----------------------------------------------------");
 Serial.println("Servo initialized. Ready to move!");
 Serial.println("Enter a speed between 1 and 50 to adjust the servo speed.");
 Serial.println("Enter a target angle (0-180) to move the servo.");
 Serial.println("Reset in order to change speed.");
 Serial.println("----------------------------------------------------");
}
```

```
void loop() {
 // Non-blocking servo movement
 moveServo();

 // Handle user input to adjust speed or set a new target angle
 handleInput();
}

// Function to move the servo to the target angle non-blockingly
void moveServo() {
 // Check if it's time to move the servo
 if (millis() - lastMoveTime >= servoSpeed) {
   lastMoveTime = millis();

   // Move the servo one step closer to the target angle
   if (currentAngle < targetAngle) {
     currentAngle++;
     myservo.write(currentAngle);
   } else if (currentAngle > targetAngle) {
     currentAngle--;
     myservo.write(currentAngle);
   }
 }
}

// Function to handle user input
void handleInput() {
 // Check if Serial data is available
 if (Serial.available()) {
   String input = Serial.readStringUntil('\n'); // Read input until newline character
   input.trim(); // Remove any leading/trailing whitespace

   if (isValidNumber(input)) {
     int value = input.toInt();

     // Check if the input is within valid servo speed range
     if (value >= 1 && value <= 50 && once) {
       servoSpeed = value;
       Serial.print("Servo speed updated to: ");
       Serial.println(servoSpeed);
       once = false;
     }
     // Check if the input is within valid angle range
     else if (value >= 0 && value <= 180) {
       targetAngle = value;
       Serial.print("Target angle updated to: ");
       Serial.println(targetAngle);
     } else {
       Serial.println("Invalid input. Enter a speed (1-50) or angle (0-180).");
     }
   } else {
     Serial.println("Invalid input. Enter a numeric value.");
   }
 }
}

// Function to check if a string contains only numeric characters
bool isValidNumber(String str) {
 for (unsigned int i = 0; i < str.length(); i++) {
   if (!isDigit(str[i])) {
     return false; // Return false if a non-digit character is found
   }
 }
 return true; // Return true if all characters are digits
}
```

Explanation of Behavior:

1. **Servo Movement Speed:**
   - The `servoSpeed` value controls how often the servo steps toward its target angle. A smaller value results in faster movement.
2. **Angle Targeting:**
   - The `targetAngle` allows the servo to move precisely to the desired position, step by step.
3. **Task Scheduling:**
   - By using `millis()`, the servo movement coexists with other operations in the loop, such as Serial communication, without interruptions.