HOMEWORK 3

(Software Engineering CS487)

Design a component which validates input values.

1. Show pseudo-code for a module which must assess an integer against a valid range. It must return a valid value as close as possible to the input and an indicator of the returned value's reliability. (2 pts)

```
function validateInteger(input, minValue, maxValue):
    // Check if input is an integer
    if not isInteger(input):
        return { value: minValue, reliability: "low" }  // Default to min value if not an integer

    // Check if input is within the valid range
    if input < minValue:
        // Value is too low
        return { value: minValue, reliability: "low" }  // Return min value with low reliability
    else if input > maxValue:
        // Value is too high
        return { value: maxValue, reliability: "low" }  // Return max value with low reliability
    else:
        // Value is within the valid range
        return { value: input, reliability: "high" }  // Return original value with high reliability
```

2. Make it reusable – identify the code segments that can remain as is and those which need to be modified such that the component could be used to process other data types (e.g., date and string values) (1 pt)

➤ To make the code reusable for processing different data types like dates and strings, we must first determine which portions may be left unmodified and which must be modified. Here's the analysis:

Segments that can be left unchanged:

• The fundamental structure and flow control of the function.
• The concept of determining if an input is inside a specified range.
• The logic for determining reliability depends on whether the input falls within a valid range.

Segments that require modification:

• The validation check to determine whether the input is an integer must be modified to support various data types.
• The algorithm for assessing whether the input is inside the valid range may need to be modified based on the data type.

<u>pseudo-code</u>

```
function validateValue(input, minValue, maxValue, dataType):

  // Check if input is of the specified data type

  if not isValidType(input, dataType):

    // If not of the specified data type, return a default value with low reliability

    return {

      value: defaultValue(dataType),

      reliability: "low"

    }


  // Check if input is within the valid range

  if input < minValue:

    // If input is below the valid range, return minimum value with low reliability

    return {

      value: minValue,

      reliability: "low"

    }

  else if input > maxValue:

    // If input is above the valid range, return maximum value with low reliability

    return {

      value: maxValue,
        reliability: "low"
    }
  else:
    // If input is within the valid range, return original value with high reliability
    return {
        value: input,
        reliability: "high"
    }
```

3. Use pseudo-code to show how an automated mission-critical system would manage the exception of a “less-than-perfectly-reliable” value. Alert the humans if automated processing is too risky. (2 pts)

```
try {
  // Validate input
  const validatedInput = validateInteger(input, 0, 100);

  // Check reliability
  if (validatedInput.reliability === "low") {
    // Handle unreliable input
    logError("Unreliable input detected");
    sendAlert("Unreliable input detected. Automated processing may be too risky.");
    const fallbackValue = getFallbackValue(); // Get a safe default value
    processInput(fallbackValue);
  } else {
    // Process input
    processInput(validatedInput.value);
  }
} catch (error) {
  // Handle error
  logError(error.message);
  sendAlert("Error occurred while processing input. Automated processing may be too risky.");
  const fallbackValue = getFallbackValue(); // Get a safe default value
  processInput(fallbackValue);
}
```