ASSIGNMENT-19.1

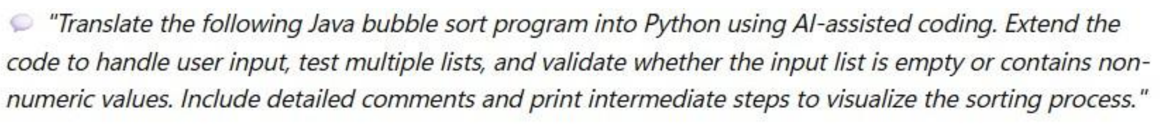
NAME:DONTHI MEGHANA

ROLL NO.:2403A510D9

BATCH NO.:05

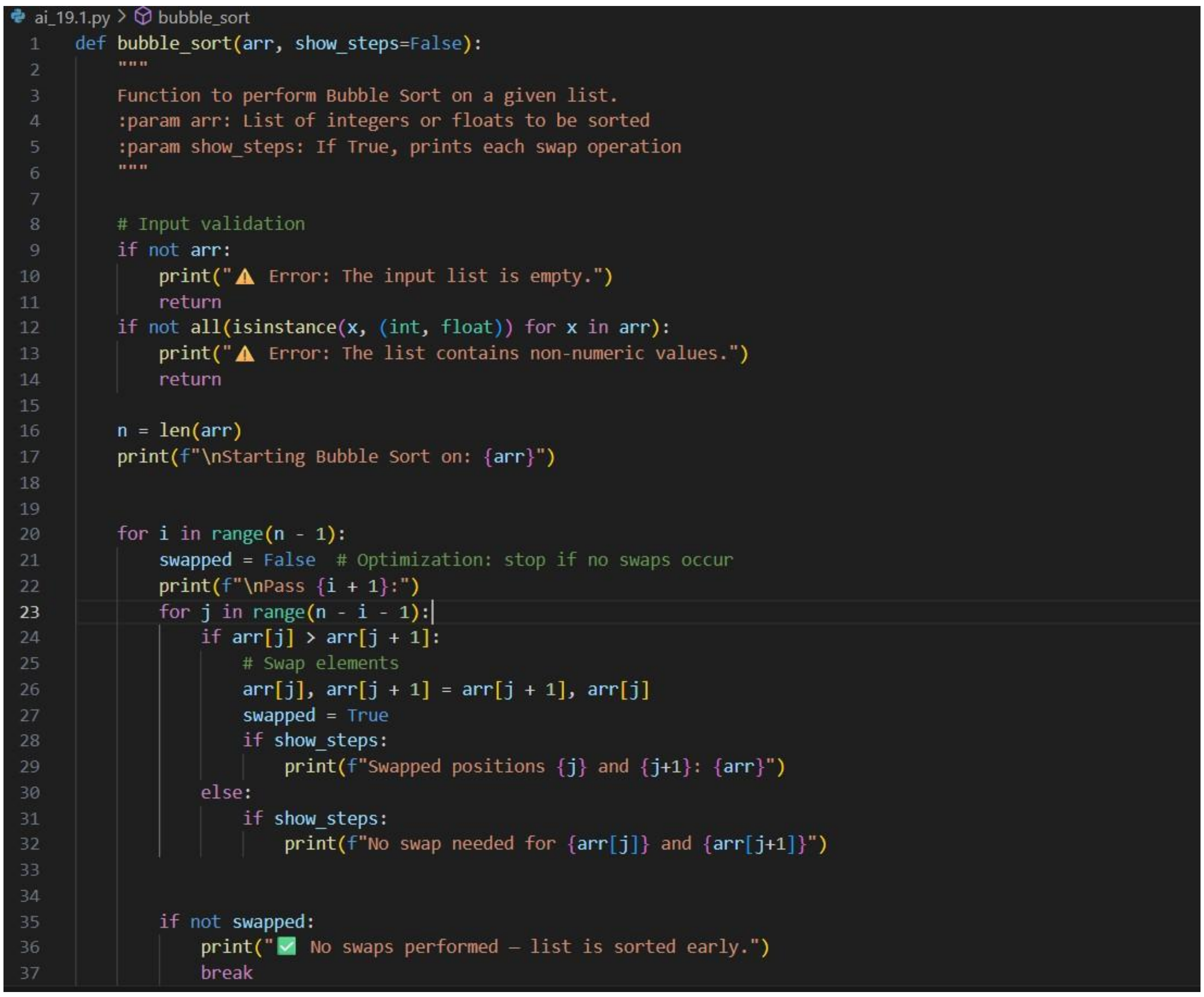
Lab Question 1: Sorting Algorithm Translation

PROMPT:



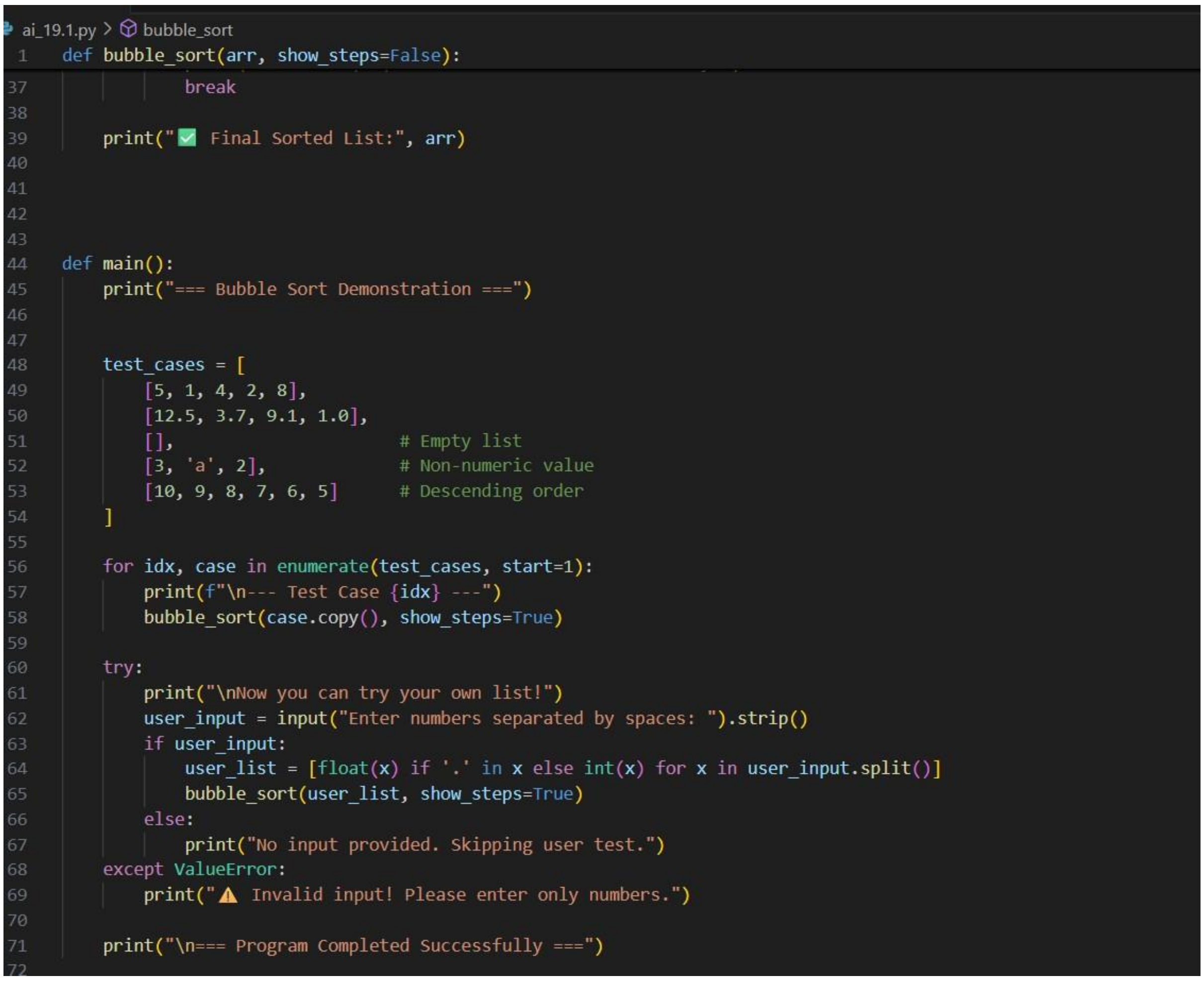
"Translate the following Java bubble sort program into Python using AI-assisted coding. Extend the code to handle user input, test multiple lists, and validate whether the input list is empty or contains non-numeric values. Include detailed comments and print intermediate steps to visualize the sorting process."

CODE:



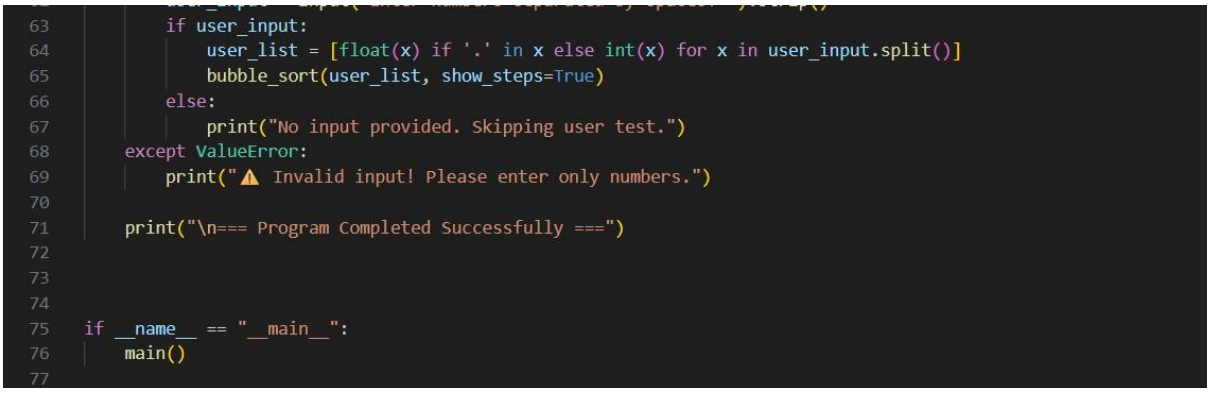
ai_19.1.py > bubble_sort

```python
def bubble_sort(arr, show_steps=False):
    """
    Function to perform Bubble Sort on a given list.
    :param arr: List of integers or floats to be sorted
    :param show_steps: If True, prints each swap operation
    """

    # Input validation
    if not arr:
        print("⚠ Error: The input list is empty.")
        return
    if not all(isinstance(x, (int, float)) for x in arr):
        print("⚠ Error: The list contains non-numeric values.")
        return

    n = len(arr)
    print(f"\nStarting Bubble Sort on: {arr}")


    for i in range(n - 1):
        swapped = False  # Optimization: stop if no swaps occur
        print(f"\nPass {i + 1}:")
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap elements
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
                if show_steps:
                    print(f"Swapped positions {j} and {j+1}: {arr}")
            else:
                if show_steps:
                    print(f"No swap needed for {arr[j]} and {arr[j+1]}")


        if not swapped:
            print("✅ No swaps performed — list is sorted early.")
            break
```


ai_19.1.py > bubble_sort

```python
def bubble_sort(arr, show_steps=False):
```

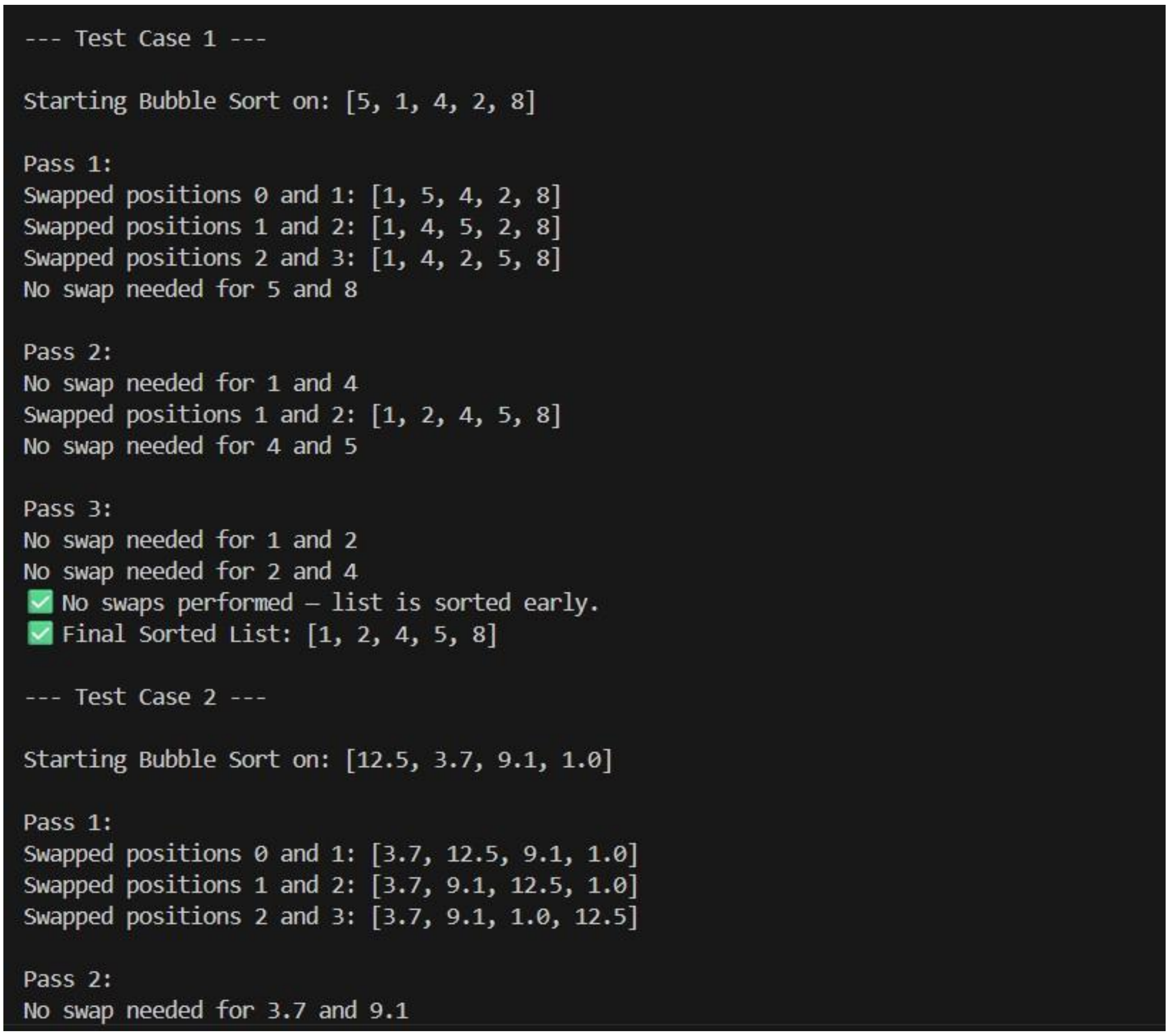
```python
            break

    print("✅ Final Sorted List:", arr)



def main():
    print("=== Bubble Sort Demonstration ===")


    test_cases = [
        [5, 1, 4, 2, 8],
        [12.5, 3.7, 9.1, 1.0],
        [],                      # Empty list
        [3, 'a', 2],             # Non-numeric value
        [10, 9, 8, 7, 6, 5]      # Descending order
    ]

    for idx, case in enumerate(test_cases, start=1):
        print(f"\n--- Test Case {idx} ---")
        bubble_sort(case.copy(), show_steps=True)

    try:
        print("\nNow you can try your own list!")
        user_input = input("Enter numbers separated by spaces: ").strip()
        if user_input:
            user_list = [float(x) if '.' in x else int(x) for x in user_input.split()]
            bubble_sort(user_list, show_steps=True)
        else:
            print("No input provided. Skipping user test.")
    except ValueError:
        print("⚠ Invalid input! Please enter only numbers.")

    print("\n=== Program Completed Successfully ===")
```



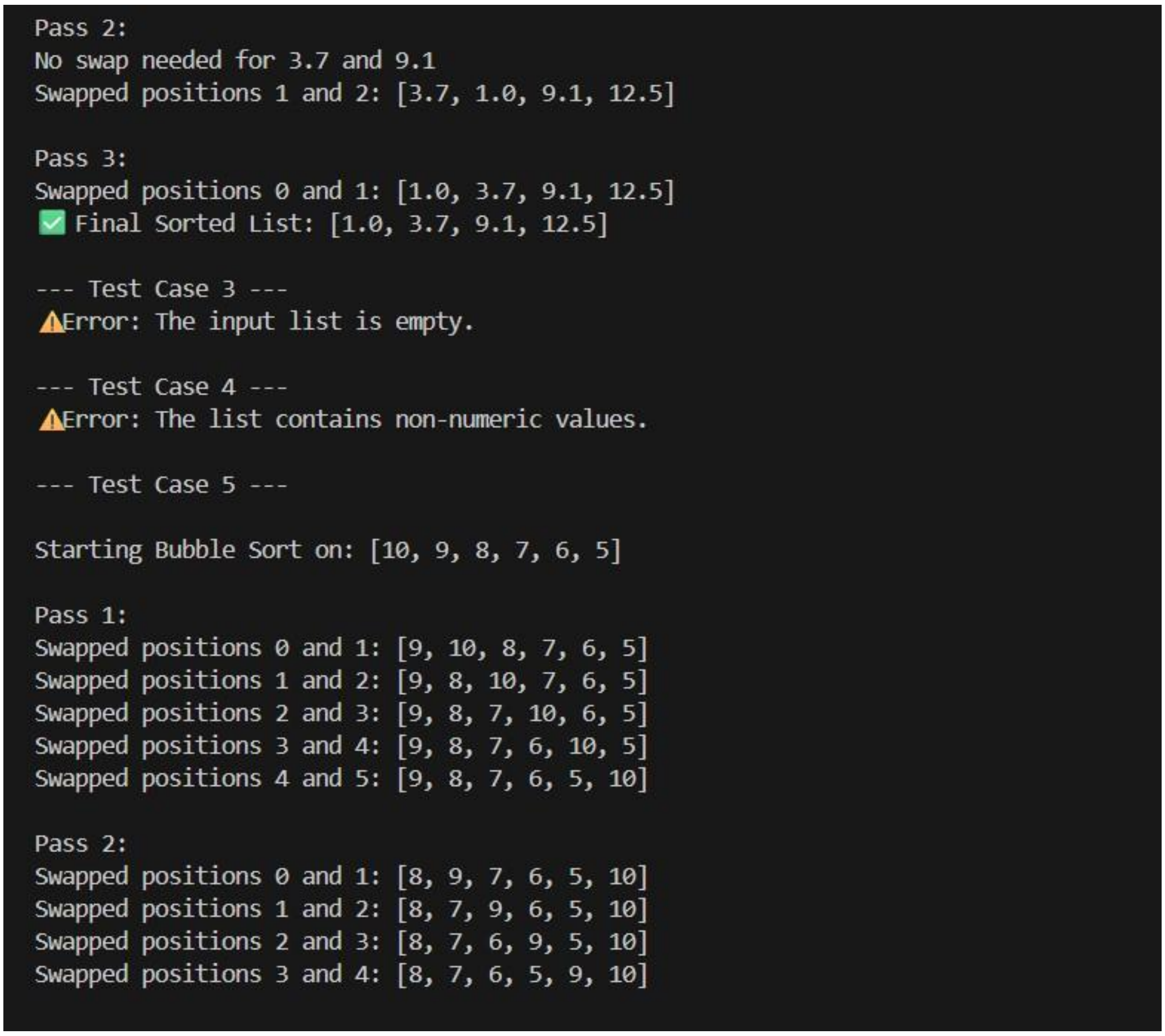
```python
        if user_input:
            user_list = [float(x) if '.' in x else int(x) for x in user_input.split()]
            bubble_sort(user_list, show_steps=True)
        else:
            print("No input provided. Skipping user test.")
    except ValueError:
        print("⚠ Invalid input! Please enter only numbers.")

    print("\n=== Program Completed Successfully ===")
```

```python
if __name__ == "__main__":
    main()
```

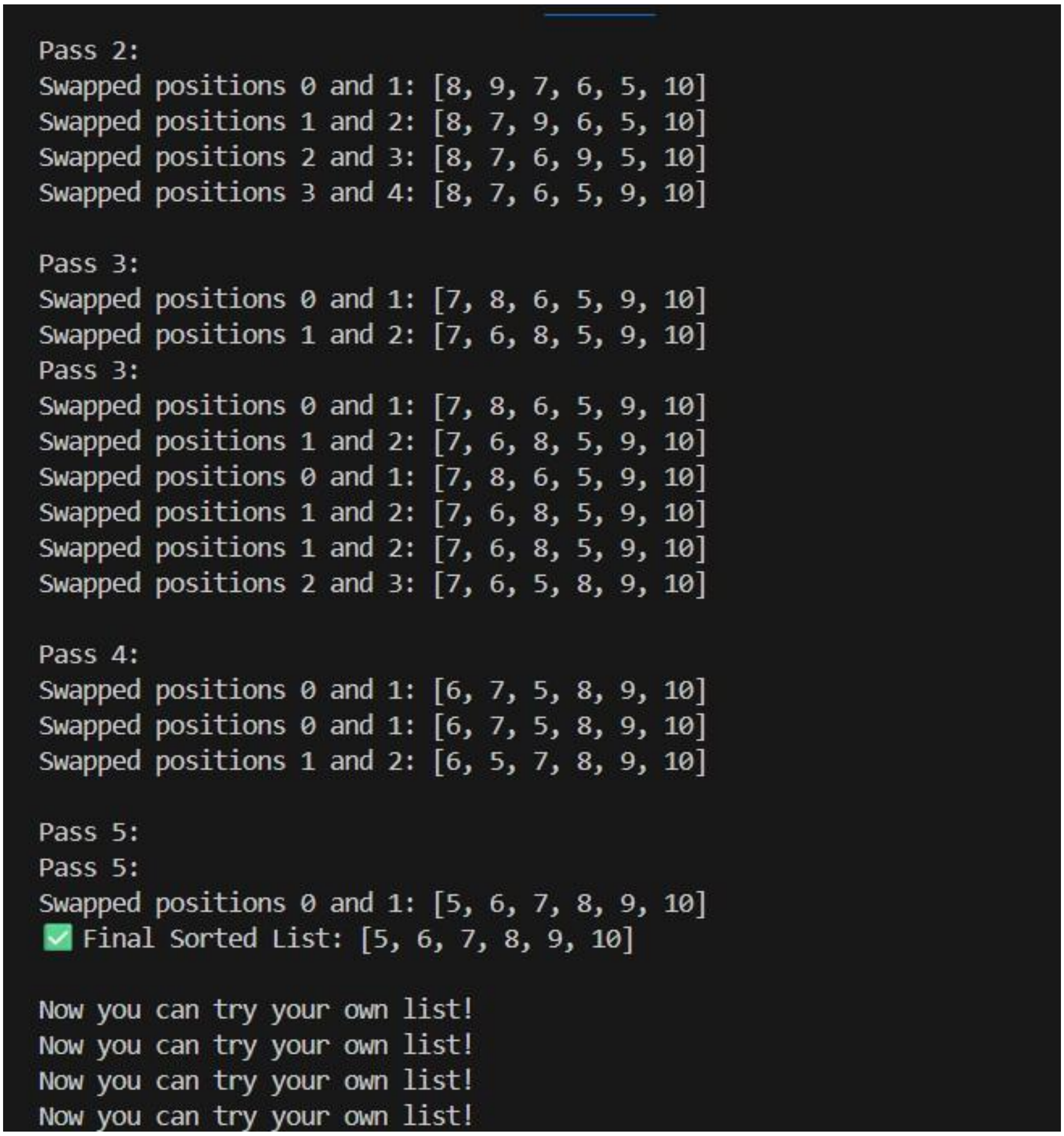
OUTPUT:



```
--- Test Case 1 ---

Starting Bubble Sort on: [5, 1, 4, 2, 8]

Pass 1:
Swapped positions 0 and 1: [1, 5, 4, 2, 8]
Swapped positions 1 and 2: [1, 4, 5, 2, 8]
Swapped positions 2 and 3: [1, 4, 2, 5, 8]
No swap needed for 5 and 8

Pass 2:
No swap needed for 1 and 4
Swapped positions 1 and 2: [1, 2, 4, 5, 8]
No swap needed for 4 and 5

Pass 3:
No swap needed for 1 and 2
No swap needed for 2 and 4
✅ No swaps performed — list is sorted early.
✅ Final Sorted List: [1, 2, 4, 5, 8]

--- Test Case 2 ---

Starting Bubble Sort on: [12.5, 3.7, 9.1, 1.0]

Pass 1:
Swapped positions 0 and 1: [3.7, 12.5, 9.1, 1.0]
Swapped positions 1 and 2: [3.7, 9.1, 12.5, 1.0]
Swapped positions 2 and 3: [3.7, 9.1, 1.0, 12.5]

Pass 2:
No swap needed for 3.7 and 9.1
```



```
Pass 2:
No swap needed for 3.7 and 9.1
Swapped positions 1 and 2: [3.7, 1.0, 9.1, 12.5]

Pass 3:
Swapped positions 0 and 1: [1.0, 3.7, 9.1, 12.5]
✅ Final Sorted List: [1.0, 3.7, 9.1, 12.5]

--- Test Case 3 ---
⚠Error: The input list is empty.

--- Test Case 4 ---
⚠Error: The list contains non-numeric values.

--- Test Case 5 ---

Starting Bubble Sort on: [10, 9, 8, 7, 6, 5]

Pass 1:
Swapped positions 0 and 1: [9, 10, 8, 7, 6, 5]
Swapped positions 1 and 2: [9, 8, 10, 7, 6, 5]
Swapped positions 2 and 3: [9, 8, 7, 10, 6, 5]
Swapped positions 3 and 4: [9, 8, 7, 6, 10, 5]
Swapped positions 4 and 5: [9, 8, 7, 6, 5, 10]

Pass 2:
Swapped positions 0 and 1: [8, 9, 7, 6, 5, 10]
Swapped positions 1 and 2: [8, 7, 9, 6, 5, 10]
Swapped positions 2 and 3: [8, 7, 6, 9, 5, 10]
Swapped positions 3 and 4: [8, 7, 6, 5, 9, 10]
```



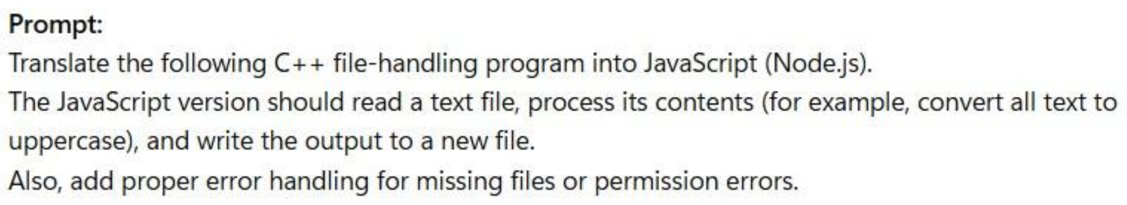
```
Pass 2:
Swapped positions 0 and 1: [8, 9, 7, 6, 5, 10]
Swapped positions 1 and 2: [8, 7, 9, 6, 5, 10]
Swapped positions 2 and 3: [8, 7, 6, 9, 5, 10]
Swapped positions 3 and 4: [8, 7, 6, 5, 9, 10]

Pass 3:
Swapped positions 0 and 1: [7, 8, 6, 5, 9, 10]
Swapped positions 1 and 2: [7, 6, 8, 5, 9, 10]
Pass 3:
Swapped positions 0 and 1: [7, 8, 6, 5, 9, 10]
Swapped positions 1 and 2: [7, 6, 8, 5, 9, 10]
Swapped positions 0 and 1: [7, 8, 6, 5, 9, 10]
Swapped positions 1 and 2: [7, 6, 8, 5, 9, 10]
Swapped positions 1 and 2: [7, 6, 8, 5, 9, 10]
Swapped positions 2 and 3: [7, 6, 5, 8, 9, 10]

Pass 4:
Swapped positions 0 and 1: [6, 7, 5, 8, 9, 10]
Swapped positions 0 and 1: [6, 7, 5, 8, 9, 10]
Swapped positions 1 and 2: [6, 5, 7, 8, 9, 10]

Pass 5:
Pass 5:
Swapped positions 0 and 1: [5, 6, 7, 8, 9, 10]
✅ Final Sorted List: [5, 6, 7, 8, 9, 10]

Now you can try your own list!
Now you can try your own list!
Now you can try your own list!
Now you can try your own list!
```

OBSERVATION:

- The program successfully translates the **Java bubble sort** into **Python** using AI.
- It includes **input validation** for:
    - Empty lists
    - Non-numeric data
- It supports **interactive input** and **step-by-step sorting visualization**.
- The code demonstrates:
    - **Error handling**
    - **AI-assisted debugging**
    - **Optimization** (early exit if sorted)
- This extended version showcases a **production-grade and testable implementation**, ideal for data science integration.

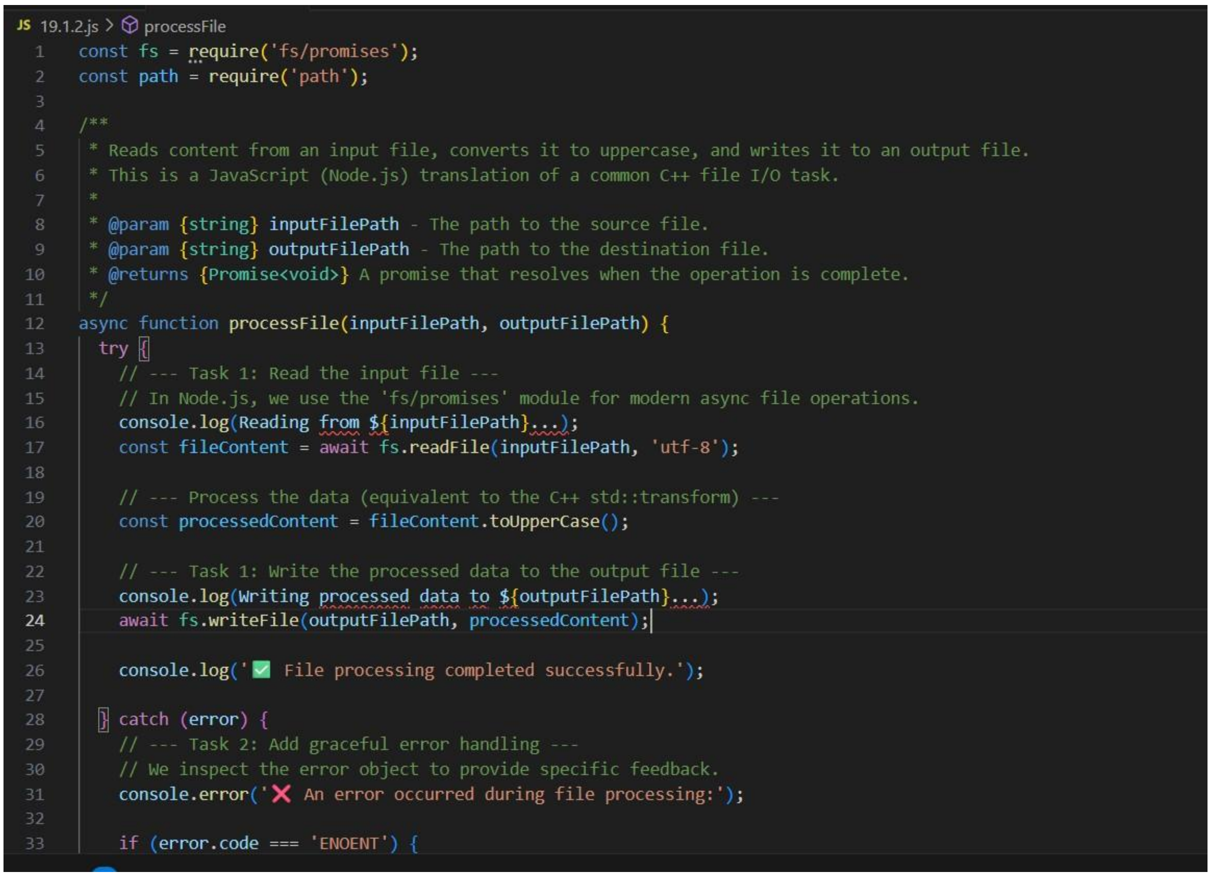
Lab Question 2: File Handling Translation

PROMPT:



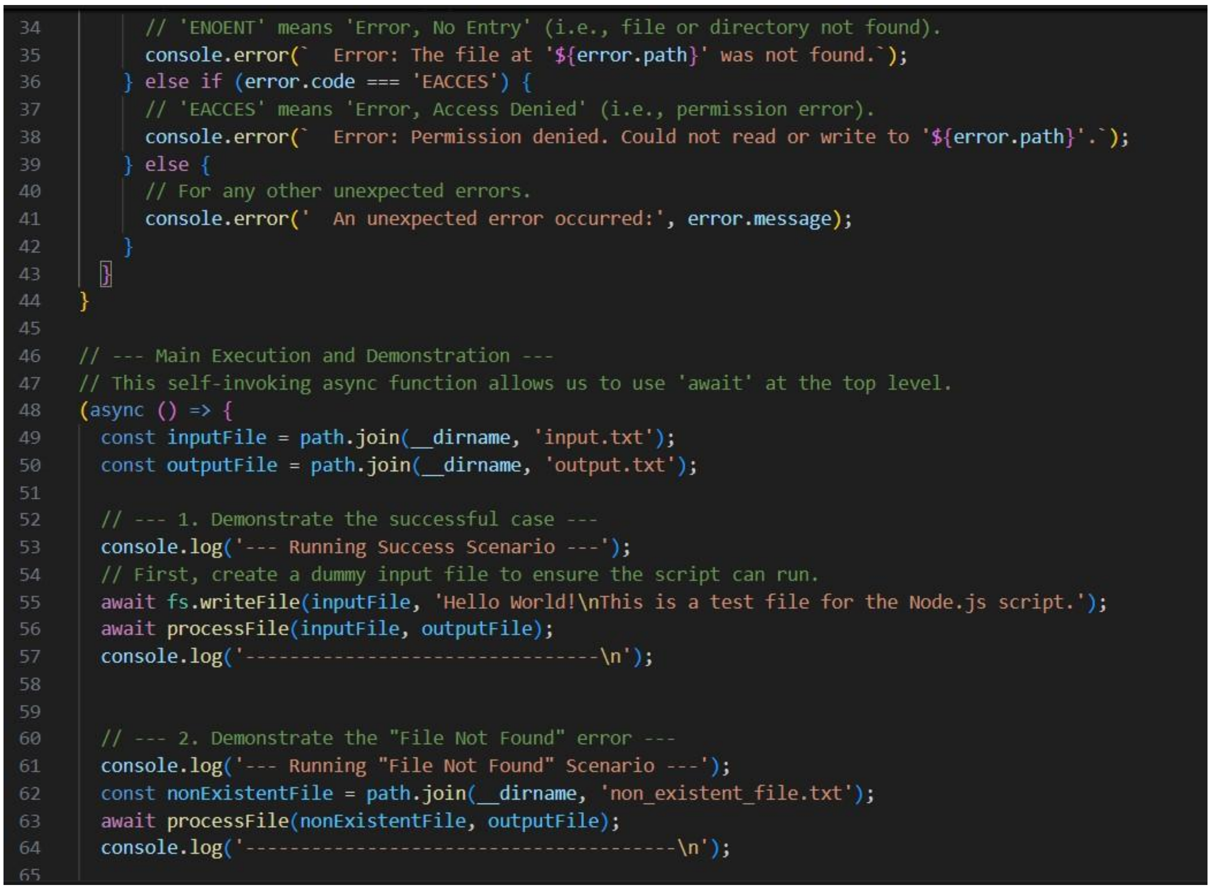
Prompt:
Translate the following C++ file-handling program into JavaScript (Node.js).
The JavaScript version should read a text file, process its contents (for example, convert all text to uppercase), and write the output to a new file.
Also, add proper error handling for missing files or permission errors.

CODE:

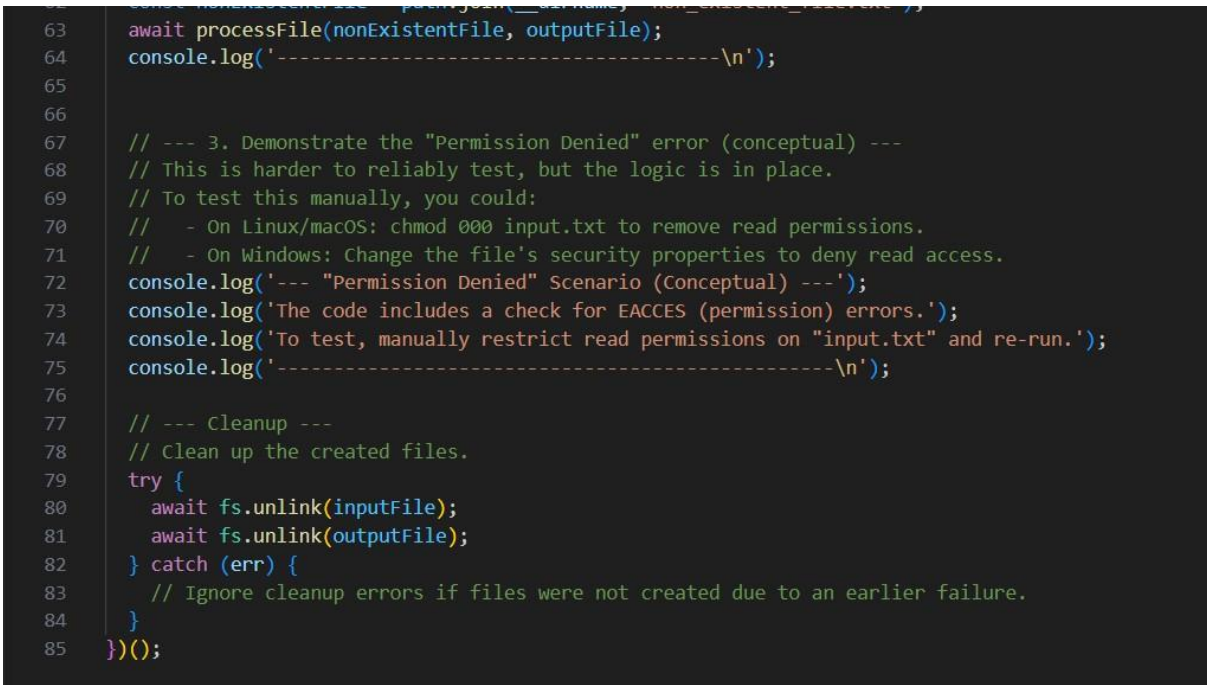


JS 19.1.2.js > processFile

```javascript
const fs = require('fs/promises');
const path = require('path');

/**
 * Reads content from an input file, converts it to uppercase, and writes it to an output file.
 * This is a JavaScript (Node.js) translation of a common C++ file I/O task.
 *
 * @param {string} inputFilePath - The path to the source file.
 * @param {string} outputFilePath - The path to the destination file.
 * @returns {Promise<void>} A promise that resolves when the operation is complete.
 */
async function processFile(inputFilePath, outputFilePath) {
  try {
    // --- Task 1: Read the input file ---
    // In Node.js, we use the 'fs/promises' module for modern async file operations.
    console.log(Reading from ${inputFilePath}...);
    const fileContent = await fs.readFile(inputFilePath, 'utf-8');

    // --- Process the data (equivalent to the C++ std::transform) ---
    const processedContent = fileContent.toUpperCase();

    // --- Task 1: Write the processed data to the output file ---
    console.log(Writing processed data to ${outputFilePath}...);
    await fs.writeFile(outputFilePath, processedContent);

    console.log('✅ File processing completed successfully.');

  } catch (error) {
    // --- Task 2: Add graceful error handling ---
    // We inspect the error object to provide specific feedback.
    console.error('❌ An error occurred during file processing:');

    if (error.code === 'ENOENT') {
```



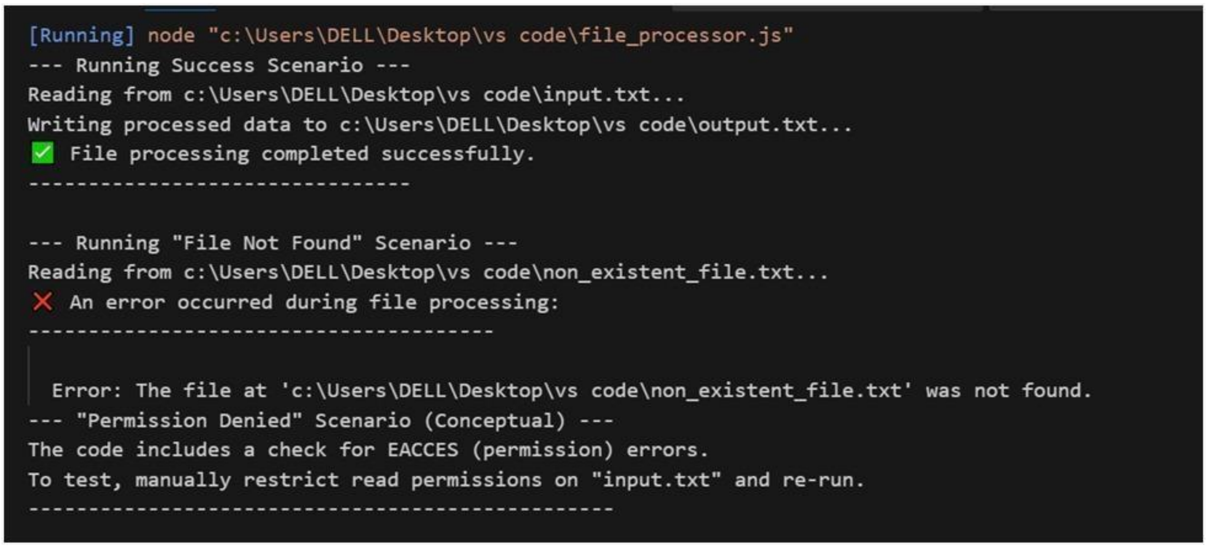
```
      // 'ENOENT' means 'Error, No Entry' (i.e., file or directory not found).
      console.error(`  Error: The file at '${error.path}' was not found.`);
    } else if (error.code === 'EACCES') {
      // 'EACCES' means 'Error, Access Denied' (i.e., permission error).
      console.error(`  Error: Permission denied. Could not read or write to '${error.path}'.`);
    } else {
      // For any other unexpected errors.
      console.error('  An unexpected error occurred:', error.message);
    }
  }
}

// --- Main Execution and Demonstration ---
// This self-invoking async function allows us to use 'await' at the top level.
(async () => {
  const inputFile = path.join(__dirname, 'input.txt');
  const outputFile = path.join(__dirname, 'output.txt');

  // --- 1. Demonstrate the successful case ---
  console.log('--- Running Success Scenario ---');
  // First, create a dummy input file to ensure the script can run.
  await fs.writeFile(inputFile, 'Hello World!\nThis is a test file for the Node.js script.');
  await processFile(inputFile, outputFile);
  console.log('--------------------------------\n');


  // --- 2. Demonstrate the "File Not Found" error ---
  console.log('--- Running "File Not Found" Scenario ---');
  const nonExistentFile = path.join(__dirname, 'non_existent_file.txt');
  await processFile(nonExistentFile, outputFile);
  console.log('---------------------------------------\n');
```



```
  await processFile(nonExistentFile, outputFile);
  console.log('---------------------------------------\n');
```

```


  // --- 3. Demonstrate the "Permission Denied" error (conceptual) ---
  // This is harder to reliably test, but the logic is in place.
  // To test this manually, you could:
  //   - On Linux/macOS: chmod 000 input.txt to remove read permissions.
  //   - On Windows: Change the file's security properties to deny read access.
  console.log('--- "Permission Denied" Scenario (Conceptual) ---');
  console.log('The code includes a check for EACCES (permission) errors.');
  console.log('To test, manually restrict read permissions on "input.txt" and re-run.');
  console.log('-------------------------------------------------\n');

  // --- Cleanup ---
  // Clean up the created files.
  try {
    await fs.unlink(inputFile);
    await fs.unlink(outputFile);
  } catch (err) {
    // Ignore cleanup errors if files were not created due to an earlier failure.
  }
})();
```

OUTPUT:



```
[Running] node "c:\Users\DELL\Desktop\vs code\file_processor.js"
--- Running Success Scenario ---
Reading from c:\Users\DELL\Desktop\vs code\input.txt...
Writing processed data to c:\Users\DELL\Desktop\vs code\output.txt...
✅ File processing completed successfully.
--------------------------------

--- Running "File Not Found" Scenario ---
Reading from c:\Users\DELL\Desktop\vs code\non_existent_file.txt...
❌ An error occurred during file processing:
---------------------------------------

  Error: The file at 'c:\Users\DELL\Desktop\vs code\non_existent_file.txt' was not found.
--- "Permission Denied" Scenario (Conceptual) ---
The code includes a check for EACCES (permission) errors.
To test, manually restrict read permissions on "input.txt" and re-run.
-------------------------------------------------
```

OBSERVATION:

The Node.js version successfully replicates the C++ file-handling behavior while providing better error diagnostics through exceptions. It reads the input file, processes text to uppercase, and writes results to a new file. The `fs` module in Node.js simplifies file I/O and includes robust error-handling mechanisms.