**Advance Programming**

**CSCI 251**

**Assignment 3**

**Name: Jeslyn Ho Ka Yan**

**SIM ID/ UOW ID: 10241485/ 8535383**

**Tutorial Group Number: Group 6**

**Table of Content**

1 Interpreted Requirements of the Program 3

2 Diagram / Illustrations of Program Design 4

3 Summary of Implementation of Each Module 5

4 Reflections on Program Development 6

# Interpreted Requirements of the Program

The purpose of this program is to process and manage geometric objects, specifically points and lines in both 2D and 3D space. The program allows users to:

* Load and store geometric data from a file.
* Apply filters to view specific types of objects (Point2D, Point3D, Line2D, Line3D).
* Sort objects based on various criteria (e.g., x-coordinate, distance from origin, line length).
* Allow ordering in ascending or descending sequence.
* Save the sorted and filtered data into an output file.

The program provides an interactive menu-driven interface to facilitate these functionalities.

# Diagram / Illustrations of Program Design

The program is designed using an **object-oriented approach**, where each geometric entity (Point2D, Point3D, Line2D, Line3D) is encapsulated in its respective class. The system is modular, with a clear separation of concerns for ease of maintenance and scalability.

**Component Breakdown:**

1. **Point2D & Point3D Classes:**
   * Represent individual points in space with coordinate values.
   * The **Point3D class** extends **Point2D**, adding a z-coordinate.
2. **Line2D & Line3D Classes:**
   * Represent line segments connecting two points.
   * Compute the **Euclidean distance** as their length.
   * **Line3D extends Line2D** by handling additional z-coordinates.
3. **Utility Functions & Templates:**
   * Generic functions for operations like **equality checking** and **scalar differences**.
   * **Templates** provide flexibility for different object types.
4. **Main Application (Menu System):**
   * **Handles user input** and **invokes corresponding functionalities** such as loading data, sorting, filtering, and saving results.
   * Ensures a **smooth workflow** by directing user choices to the correct function calls.

**Program Flow:**

1. **Program Start:** The user is presented with a menu.
2. **Data Loading:** If not already loaded, the user provides an input file.
3. **Filtering Selection:** The user selects whether to view Point2D, Point3D, Line2D, or Line3D objects.
4. **Sorting Preferences:** The user specifies sorting criteria (e.g., x-coordinate, length).
5. **Sorting Order:** The user chooses ascending or descending order.
6. **Processed Data Display:** The program applies the filters and sorting, then presents the results.
7. **Data Storage:** The user can choose to save the filtered and sorted data.
8. **Program Exit:** The program continues until the user opts to exit.

# Summary of Implementation of Each Module

**Point2D Class**

* **Attributes:** x, y coordinates.
* **Methods:**
  + Computes distance from the origin using the Euclidean formula.
  + Overloaded comparison operators for sorting and equality checks.
  + Getter methods for x, y, and distance from the origin.

**Point3D Class**

* Inherits from Point2D and adds a z-coordinate.
* Overrides the distance computation to consider the third dimension.
* Implements sorting based on x, y, and z values.

**Line2D Class**

* Defines a line segment using two Point2D objects as endpoints.
* Computes the length using the Euclidean formula.
* Implements comparison operators for sorting.
* Provides getter methods for line endpoints and length.
* Includes static comparison functions for ascending and descending sorting.

**Line3D Class**

* Extends Line2D to work in 3D space.
* Computes line length using the 3D Euclidean distance formula.
* Supports sorting and equality checks.

**Template Functions (MyTemplates.h)**

* Implements generic functions for scalar differences and equality checks.
* Overloaded for both primitive types and geometric objects.
* Ensures code reusability across different modules.

**Main Program (csci251\_a3.cpp)**

* Implements menu-based interaction.
* Loads and processes input data.
* Calls functions for filtering, sorting, and viewing data.
* Supports saving results to a file.
* Manages program flow, ensuring structured execution.

# Reflections on Program Development

**Assumptions Made**

* Input files are formatted correctly with consistent data structures.
* Users will provide valid filenames when prompted.
* Sorting will be applied only after filtering is specified.

**Difficulties Faced**

* Parsing data from the input file, especially for handling different object types.
* Ensuring template functions worked correctly with both primitive and object types.
* Implementing a robust sorting mechanism while maintaining efficiency.

**What Could Have Been Done Better**

* Improved error handling for input validation and file operations.
* More comprehensive testing for edge cases.
* Optimization of sorting operations to reduce redundant comparisons.

**Possible Enhancements in the Future**

* GUI implementation for an improved user experience.
* Additional geometric objects such as circles or polygons.
* Exporting results to multiple formats (CSV, JSON).
* Integration with a database for better data management.

**What Have You Learned**

* The importance of modular programming and object-oriented design.
* How to use templates effectively for generic programming.
* The impact of sorting algorithms on performance.
* Best practices for user input handling in a console application.