Interpreted requirements of the program

1. Create and develop 4 classes
   1. Point2D class
   2. Point3D class
   3. Line2D class
   4. Line3D class
2. Read in data from a user-specified filename
   1. Create my own input manipulator(s) to read in data
   2. Input file contains data meant for different kinds of classes
   3. Overload the extraction operator for each of the 4 classes
3. Use a STL container to store all various objects from the 4 classes
   1. Remove all duplicate rows of data
4. Display the data
   1. According to the given format
   2. In different orders of each attribute
   3. Overload the insertion operator for each of the 4 classes
5. Develop 5 generic template functions and save them in ‘MyTemplates.h’
   1. difference
   2. scalar\_difference
   3. lessThan
   4. greaterThan
   5. equals
6. Allow user to specify
   1. filtering criteria
   2. sorting criteria
   3. ascending or descending order
7. Store the records displayed in a user-specific filename

Diagram / Illustrations of program design

I have chosen to use vectors to store all objects of the 4 classes

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Point2D  Object | Point2D  Object | Point2D  Object | Point2D  Object | Point2D  Object | Point2D  Object | Point2D  Object |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Point3D  Object | Point3D  Object | Point3D  Object | Point3D  Object | Point3D  Object | Point3D  Object | Point3D  Object |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Line2D  Object | Line2D  Object | Line2D  Object | Line2D  Object | Line2D  Object | Line2D  Object | Line2D  Object |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Line3D  Object | Line3D  Object | Line3D  Object | Line3D  Object | Line3D  Object | Line3D  Object | Line3D  Object |  |  |

Summary of implementation of each module in my program

* Read in data
  + Prompts the user for name of input file
  + Informs the user if file not found or else
  + For each line
    - reads in the type of data and creates the respective object using the data read
    - Checks if a similar object already exist within its respective container
      * If it exist then data is not added to the container
      * If it does not exit then data is added to the container
    - Repeats until end of file (or until error occurs when reading)
* Specify filtering criteria
  + Shows the current filtering criteria (default is Point2D)
  + Allows the user to choose which objects to be displayed
    - Point2D, Point3D, Line2D or Line3D
  + If user chooses Point2D or Point3D, sorting criteria will be ‘x-ordinate’ by default
  + If user chooses Line2D or Line3D, sorting criteria will be ‘Pt. 1’ by default
* Specify sorting criteria
  + Shows the current sorting criteria (default is x-ordinate)
  + Allows the user to choose which attribute of the objects to be sorted by
    - For Point2D, options are x, y ordinates and distance from origin
    - For Point3D, options are x, y, z ordinates and distance from origin
    - For Line2D and Line3D, options are Pt. 1, Pt. 2 and length value
* Specifying sorting order
  + Shows the current sorting order
  + Allows the user to choose ascending or descending order
* View data
  + Displays the data according to the filter and sorting criteria and its sorting order
  + Format is as mentioned in the requirements
* Store data
  + Stores the data as displayed to a file
  + Alerts the user if unable to create the file
* Create a random data text file (eg. messy.txt)
  + Generates a random file for input
* Delete an object
  + Prompts the user which type of object to delete
  + Alerts user if the container of the type is already empty or else
  + If type chosen is Point2D or Point3D
    - Prompts the user for the x and y coordinates (and z ordinate for Point3D)
    - Checks if values are within the valid range
    - Alerts the user if the object is not found or else deletes the object from the container
  + If type chosen is Line2D or Line3D
    - Prompts the user for the x and y coordinates (and z ordinate for Line3D) for Pt. 1 and Pt. 2
    - Checks if values are within the valid range
    - Alerts the user if the object is not found or else deletes the object from the container
* Add an object
  + Prompts the user which type of object to delete
  + If type chosen is Point2D or Point3D
    - Prompts the user for the x and y coordinates (and z ordinate for Point3D)
    - Checks if values are within the valid range
    - Adds the object to the container
  + If type chosen is Line2D or Line3D
    - Prompts the user for the x and y coordinates (and z ordinate for Line3D) for Pt. 1 and Pt. 2
    - Checks if values are within the valid range
    - Adds the object to the container
* Show statistical information
  + Displays the maximum, minimum and average values of the distance from origin of Point2D and Point3D objects
  + Displays the maximum, minimum and average values of the length of Line2D and Line3D objects
* Exit program
  + Exits the program with and tells the user “Have a nice day!”

Reflections on program development

**Difficulties faced**

In the beginning I was faced with the dilemma of choosing a stl container. Initially I wanted to follow from what I did in Assignment 2, which was using a map to store the objects. This is because with maps, we could easily solve the hassle of sorting by different attributes. However later on I felt that since we need to sort the objects by each of their attributes (assuming no additional attributes are added) we would need to implement many indexes in other words, many maps. This would mean that much memory space would be needed because as each map element (pair) stores an attribute of the object (eg. x of Point2D) at the ‘first’ location of the pair (and assuming I store the object pointer at ‘second’ instead of the object itself in order to save memory), we would need to have n maps for an object that has n attributes and hence the indexing would already take up about the same memory space as the objects themselves. Also many indexes (maps) would also mean more updating to be done whenever an object is inserted in the container. Sets too is not an option as it is good when we have a fixed order for which objects are arranged, but in our case we need a flexible order of sorting. Hence the 3 STL containers left that we have learnt about are vectors, deques and lists. Viewing list as linked lists and deques as doubly linked list, I feel that sorting both of them might not be very efficient as it requires handling of the links or pointers which might take more processing. Therefore I feel that vectors was the ideal choice for this assignment.

**What could have been done better / possible enhancements in future**

Due to time constraint, there are many areas that could be enhanced further. One of such instance is that we could calculate the statistics of each attribute of the object such as the maximum, minimum and average value of x for all the Point2D objects. We could also write function to compute and print out the linear equations that describe Line2D and Line3D. Furthermore, we could also have a function to check whether the lines intersect or are parallel to each other.

**What I have learnt**

The use of template functions and input and output manipulators can greatly simplify the program and are a great tool for a programmer. For example, I have used a stringstream with its str () function to display the output the user desires. The same thing is done when I output the records into a file. In other words cout<<p2DListAsc () outputs it to the screen while outfile<<p2DListAsc () directly outputs the same thing into a file, this saves effort formatting another similar output to a file. Template functions allows the use of many different data types, in this case a function like lessThan can have parameters of each of the 4 classes, thus saving time and effort to rewrite the same code for different object types.