Assignment 3

Aim

The objectives of this assignment includes:

- Learning about generic programming templates, operator overloading, STL (containers & algorithms) and writing io manipulators
- Apply the concepts learnt by developing a data processing program

Background

In this assignment, you are required to develop a program that reads in and process some 'messy' records from a file that contains data meant for different kinds of classes. These data are jumbled up and unsorted, and to make it worse, for any particular row of record, there may be multiple duplicates scattered over the entire file!

You program should be called 'csci251_a3.**exe**', and should posses the following capabilities:

a) read in the records from a user-specified filename
b) remove all duplicate rows of data
c) filter and display the data according to user-specified sorting criteria
d) store the records displayed in c), in a user-specified filename

The next section provides information about the requirements for developing this program.

Task Requirements

A) **Appendix A** provides a sample input data from a file called 'messy.txt'. It contains information meant to be stored in 4 classes: '`Point2D`', '`Point3D`', '`Line2D`' and '`Line3D`'. Please refer to the table in **Appendix A** for a description of the format in which the input data for each of the classes is stored.

B) **Note1**: You are to research and determine which kind of STL containers (e.g. Map, Vector, Set, Lists etc) you should use, to store all the various objects from the 4 classes. For this assignment you are not allowed to use array [ ] to store any of your data!

C) **Appendix B** provides a description of the 4 classes: '`Point2D`', '`Point3D`', '`Line2D`' and '`Line3D`', and the relationships between them. You are to study the diagrams and implement them accordingly.

D) **Appendix C** provides the sample output format and a description of the format requirements, for each of the 4 classes. These format are to be applied whether the data from these classes are output to a file or terminal.

E) **Note2**: To output data, you are required to create your own output manipulator(s) to display/store data in the format described in **Appendix C**. You are further required to overload the insertion operator '**<<**', for each of the 4 classes, to support the process of inserting data to the terminal, or the relevant file stream. (Hint : Research on the concept of writing output manipulators. E.g. https://en.cppreference.com/w/cpp/io/manip/left)

F) **Note3**: All output data must not contain any duplicates! There are many approaches to solving this problem. Firstly, you could check for, and remove duplicate records at the point of reading in the input. Alternatively, you could temporarily store the data in a STL container, research and make use of any STL algorithm to search for, and remove the duplicates. Another (inefficient) way is to store everything in STL container, but your program needs to ensure that when user wishes to see / store the records in a file, no duplicate records are shown.

G) **Appendix D** provides a description of a few generic template functions that you are supposed to develop. These 'utility' functions plays a supporting role, and they should be developed in a separate header file called '**MyTemplates.h**'.

H) Your program should allow user specify the **filtering criteria** so that user can specify which set of records he wishes to view / store. Your program should allow the following options:

i) Point2D records <= default selected option
ii) Point3D records
iii) Line2D records
iv) Line3D records

I) Your program should allow user to specify the **sorting criteria** so that user can specify which attributes (of a set of records) to order the data by. The sorting criteria is based on the current filtering criteria.

For example, if the current filtering criteria is 'Only Point2D records', your program should restrict user to the options of sorting the data by 'x', 'y' & scalar value 'distFrOrigin' only!

Please refer to **Appendix E** for a detailed description of the combinations of filtering criteria, and the respective (allowable) sorting criteria.

J) **Hint:** It is not necessary to develop your own sorting algorithm, or make use of any of the classical algorithms like 'Quick-Sort', 'Bubble-Sort' to fulfill the sorting requirements. There is a function defined in STL algorithm (i.e. #include <algorithm>) called '`sort`'. You should research on its usage, code the necessary comparator functions (for each of the 4 classes + sorting criteria). Once you mastered its usage, you will easily achieve the desired sorting effect using less than 3 lines of code!

K) To assist you in visualizing the desired program interactions, please refer to **Appendix F** which provides a sample menu displaying the output data / messages in response to user input.

L) Once the program is completed and tested to be working successfully, you are highly encouraged to add on "new features" to the program that you feel are relevant to the problem. Additional marks may be awarded subject to the relevancy and correctness of the new functionalities. (Note : the additional features will only be considered IF the program has correctly fulfilled all the basic requirements elaborated in the earlier sections!)

M) You are to use only C++ language to develop your program. There is no restriction on the IDE as long as your source files can be compiled by g++ compiler (that comes packaged in Ubuntu linux) and executed in the Ubuntu terminal shell environment.

Deliverables

1) The deliverables include the following:

   a) The actual working C++ program (soft copy), ***with comments on each file, function or block of code*** to help the tutor understand its purpose.

   b) A softcopy word document that elaborates on:
      - (Interpreted) requirements of the program
      - Diagram / Illustrations of program design
      - Summary of implementation of each module in your program
      - Reflections on program development (e.g. assumptions made, difficulties faced, what could have been done better, possible enhancements in future, ***what have you learnt***, etc)

   c) A program demo/software testing during lab session. You must be prepared to perform certain tasks / answer any questions posed by the tutor.

2) IMPT: Please **follow closely**, to the submission instructions in **Appendix G**, which contains details about what to submit, file naming conventions, when to submit, where to submit, etc.

3) The software demo / testing will be held during lab session where you are supposed to submit your assignment. Some time will be allocated for you to present / demonstrate your program's capabilities during the session.

Grading

Student's deliverable will be graded according to the following criteria:

(i) Program fulfills all the basic requirements stipulated by the assignment

(ii) Successful demonstration of a working program, clarity of explanation / presentation and satisfactory answers provided during Q & A session.

(iii) Additional effort (e.g. enhancing the program with *relevant* features over and above task requirements, impressive, 'killer' presentation)

(iv) After the submission of deliverables, students will be required undergo a software testing process (to determine the correctness and fulfillment of software requirements.) Further instructions will be given by the Tutor during the subsequent respective labs. Please pay attention as failure to adhere to instructions will result in deduction of marks.

Tutor's note:
In the real working world, satisfactory completion of your tasks is no longer enough. The capability, efficiency and robustness of your system to operate under different testing conditions, and the ability to add value, communicate and/or demonstrate your ideas with clarity is just as important as correct functioning of your program. The grading criteria is set to imitate such requirements on a 'smaller scale'.

APPENDIX A

(Sample 'messy' data from an input file)

```
Point2D, [3, 2]
Line3D, [7, 12, 3], [-9, 13, 68]
Point3D, [1, 3, 8]
Line2D, [5, 7], [3, 8]
Point2D, [3, 2]
Line3D, [7, -12, 3], [9, 13, 68]
Point3D, [6, 9, 5]
Point2D, [3, 2]
Line3D, [70, -120, -3], [-29, 1, 268]
Line3D, [25, -69, -33], [-2, -41, 58]
Point3D, [6, 9, -50]
...
```

Note:

1) Some data, (e.g. '`Point2D, [3, 2]`') can be repeated multiple times (i.e. they are duplicated data), this applies to all other kinds of data as well.

2) In each line, the 1st field will contain the class's name (e.g. '`Point2D`', '`Point3D`', '`Line2D`' and '`Line3D`')

3) The delimiter separating the 1st field from the rest of the data, is a comma, followed by a space char (i.e. '**,** ')

4) The delimiter separating each number in the 2D/3D coordinate, is also a comma, followed by a space char (i.e. '**,** ')

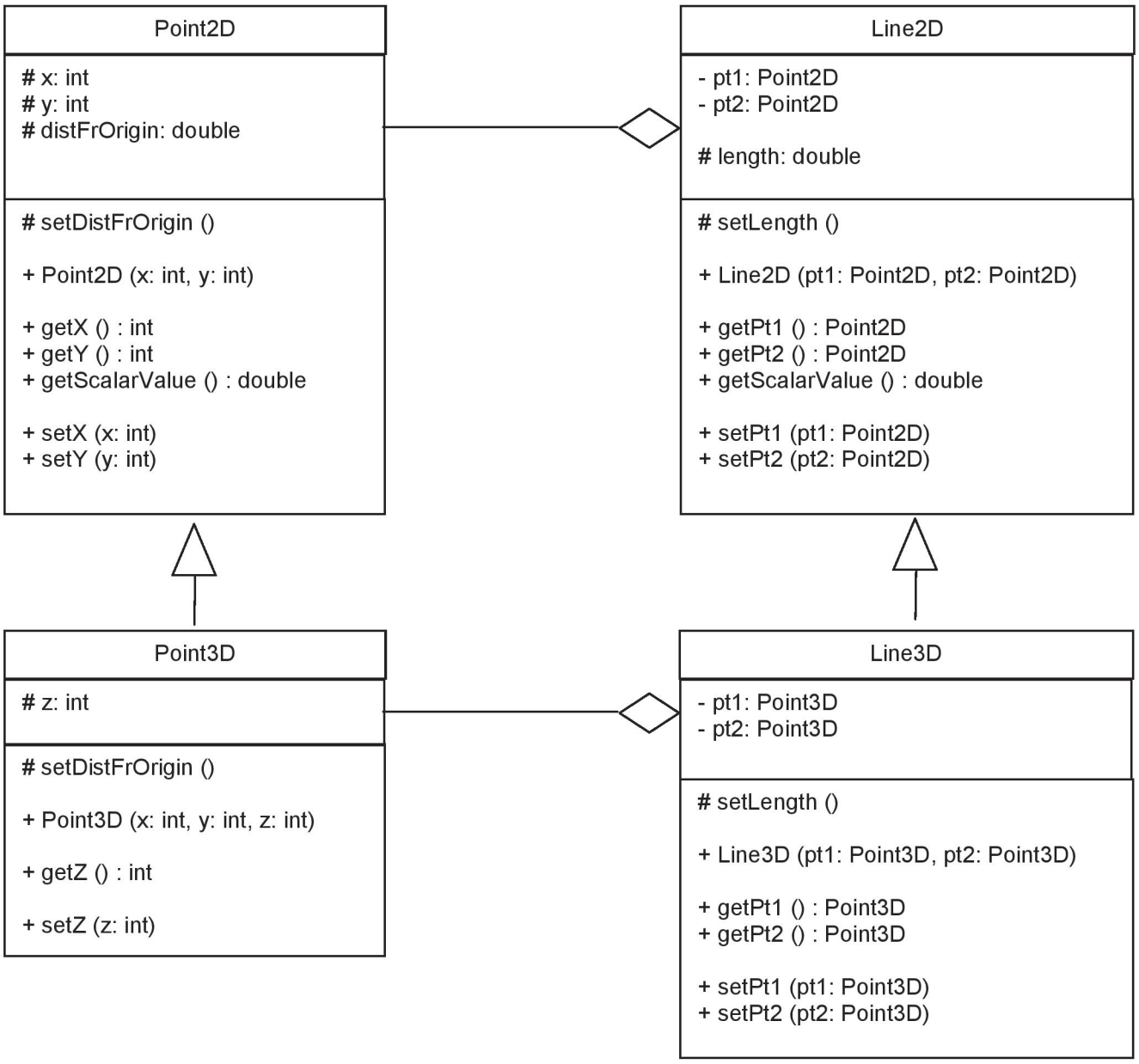
5) Each 2D/3D point's data, is enclosed by the square brackets '**[**' and '**]**'

6) Each Line2D / Line3D's data consists of two points, each enclosed by square brackets, and the delimiter separating each point is also a comma, followed by a space char (i.e. '**,** ')

7) You may assume that the range of each number in the x, y or z coordinate can be anything **from** -999 **to** 999

APPENDIX B

(The 4 classes, and their relationships)



APPENDIX B (con't)

Note:

1) The above diagrams depict the bare minimum requirements for the 4 classes whose attributes and methods you MUST implement

2) In `Point2D` class, `setDistFrOrigin ()` method computes the distance of the point to the origin (0, 0), and initializes the attribute `distFrOrigin` with this distance value. The formula to compute is as follows:

   $distFrOrigin = \sqrt{(x - 0)^2 + (y - 0)^2}$ OR

   $distFrOrigin = \sqrt{x^2 + y^2}$

   Note : √ means square root

3) In `Point2D` class, `getScalarValue ()` method is merely an accessor method that returns the value of attribute `distFrOrigin`.

4) In `Line2D` class, `setLength ()` method computes the distance between its own `Point2D` attributes `pt1` and `pt2`, and initializes the attribute `length` with this distance value. The formula to compute is as follows:

   $length = \sqrt{(pt1.x - pt2.x)^2 + (pt1.y - pt2.y)^2}$

5) In `Line2D` class, `getScalarValue ()` method is merely an accessor method that returns the value of attribute `length`.

6) In `Line3D` class, `setLength ()` method computes the distance between its own `Point3D` attributes `pt1` and `pt2`, and initializes the attribute `length` with this distance value. The formula to compute is as follows:

   $length = \sqrt{(pt1.x - pt2.x)^2 + (pt1.y - pt2.y)^2 + (pt1.z - pt2.z)^2}$

7) For all the 4 classes, you are free to declare and implement any **additional** attributes and methods like:

   - Overloading io-stream, arithmetic, comparison or any other operators '<<', '-', '==', etc.
   - Writing static comparator functions to use in STL algorithms and containers,
   - Writing io manipulators specific to a particular class
   - Writing any other supporting helper functions necessary to fulfill the requirements of this assignment.

APPENDIX C

(General Output Format for `Point2D` & `Point3D` data)

Point2D

```
    X      Y     Dist. Fr Origin
- - - - - - - - - - - - - - - - - - -
[  -9,    -9]    12.728
[ -99,   -99]    140.007
[-999,  -999]    1412.799
[   3,     3]    4.243
[  23,    23]    32.527
[ 123,   123]    173.948

. . .
```

Point3D

```
    X      Y      Z     Dist. Fr Origin
- - - - - - - - - - - - - - - - - - - -
[  -9,    -9,    -9]    15.589
[ -99,   -99,   -99]    171.473
[-999,  -999,  -999]    1730.320
[   3,     3,     3]    5.196
[  23,    23,    23]    39.837
[ 123,   123,   123]    213.042

. . .
```

Note:

1) The allocated width to store each x, y and/or z ordinate value is 4 'spaces', inclusive of minus sign

2) All x, y, z values are strictly whole numbers (i.e. integers) and all `Dist. Fr Origin` values are strictly decimal (i.e. double), with precision set at up to 3 decimal places

3) The 1st 2 lines (representing the 'header') is compulsory. The alignment of the 'X', 'Y' or 'Z' column names in the header is vertically aligned to its respective last digit's position!

4) The should be 3 'spaces' between the end of each point's data values, and its corresponding `Dist. Fr Origin` values

APPENDIX C (con't)

(General Output Format for `Line2D` & `Line3D` data)

Line2D

```
 P1-X  P1-Y     P2-X  P2-Y    Length
- - - - - - - - - - - - - - - - - - - -
[  -9,   -9]   [ -99,  -99]   127.279
[ -99,  -99]   [-999, -999]   1272.792
[-999, -999]   [  -9,   -9]   1400.071
[   3,    3]   [ 999,  999]   1408.557
[ 999,  999]   [-999, -999]   2825.599
[-999, -999]   [   3,    3]   1417.042

. . .
```

Line3D

```
 P1-X  P1-Y  P1-Z     P2-X  P2-Y  P2-Z    Length
- - - - - - - - - - - - - - - - - - - - - - - - - -
[  -9,   -9,   -9]   [ -99,  -99,  -99]   155.885
[ -99,  -99,  -99]   [-999, -999, -999]   1558.846
[-999, -999, -999]   [  -9,   -9,   -9]   1714.730
[   3,    3,    3]   [ 999,  999,  999]   1275.123
[ 999,  999,  999]   [-999, -999, -999]   3460.638
[-999, -999, -999]   [   3,    3,    3]   1735.515

. . .
```

Note:

5) In Line2D, the formatting for data under '`P1-X`', '`P1-Y`', '`P2-X`' and '`P2-Y`' headings is similar to that for Point2D. The corresponding formatting applies for the case of Line3D which is similar to that specified for Point3D.

6) The 1st 2 lines (representing the 'header') is compulsory. The alignment of the '`P1-X`', '`P1-Y`', '`P1-Z`', '`P2-X`', '`P2-Y`' and '`P2-Z`' column names in the header is vertically aligned to its respective last digit's position!

7) The should be 3 'spaces' between the end of each point's data values, and its corresponding `Length` values

APPENDIX D

(Description of Generic Function Template)

<table>
<tr><th>Template function</th><th>Parameter description</th><th>If param(s) is of the following Class / Type :</th><th>“Meaning” of the template function when applied to (param’s) Class / Type</th></tr>
<tr><td rowspan="4">Name :
scalar_
difference

Return Type :
double</td><td rowspan="4">No. of parameters : 2

Type of 1st parameter : ‘T’

Type of 2nd parameter : ‘T’</td><td>Point2D</td><td rowspan="4">The absolute value difference in the scalar values betw. parameters 1 and 2.

(Hint: make use of the method getScalarValue() mentioned in Appendix B!)</td></tr>
<tr><td>Point3D</td></tr>
<tr><td>Line2D</td></tr>
<tr><td>Line3D</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="5">equals

Return Type :
‘bool’</td><td rowspan="5">No. of parameters : 2

Type of 1st parameter : ‘T’

Type of 2nd parameter : ‘T’</td><td>Numeric primitives (int, double, etc)</td><td>parameter 1 == parameter 2</td></tr>
<tr><td>Point2D</td><td>param 1’s x == param 2’s x AND
param 1’s y == param 2’s y</td></tr>
<tr><td>Point3D</td><td>param 1’s x == param 2’s x AND
param 1’s y == param 2’s y AND
param 1’s z == param 2’s z</td></tr>
<tr><td>Line2D</td><td rowspan="2">param 1’s pt1 == param 2’s pt1
AND
param 1’s pt2 == param 2’s pt2</td></tr>
<tr><td>Line3D</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>

Note:

1) Based on the information provided in the table, you will need to overload the relevant operators in the affected classes, in order to implement the ‘meaning’ correctly, when the generic function’s algo is applied on the 4 classes

2) The above generic function templates should be implemented in a header file called ‘`MyTemplates.h`’.

APPENDIX E

(Combinations of filtering + sorting criteria)

Filtering Criteria	Sorting Criteria (allow sorting by …)
Point2D records	i) X ordinate value (default) ii) Y ordinate value iii) Dist. Fr Origin value
Point3D records	i) X ordinate value (default) ii) Y ordinate value iii) Z ordinate value iv) Dist. Fr Origin value
Line2D records Line3D records	i) X and Y coordinate values of Pt. 1 (default) ii) X and Y coordinate values of Pt. 2 iii) Length value

Note:

1) Sorting by **X and Y coordinates** is done, by first ordering all rows according to x-ordinate value, this will have a 'sorting and bunching' effect that groups rows with the same x-ordinate values together, if it exists.

   Within each 'group' with similar x-ordinate values, the sorting order (assuming it is ascending), is then applied to the y-ordinate, such that rows with the same X, but smaller Y value will be 'above' those with same X, but bigger Y values.

2) For all sorting criteria, you should allow sorting in both ASCENDING and DESCENDING order! (Assume default is 'ASC').

3) If you are using STL containers / algorithms to handle the storage of your objects and sorting, you need to implement the relevant function comparators for each of the 4 classes. These function comparators should ideally be implemented as static boolean functions under each of the 4 classes.

APPENDIX F

(Sample Menu Interactions)

```
Student ID      : 1234567
Student Name    : Tan Ah Meng Elvis
-------------------------------------------------
Welcome to Assn3 program!

1)      Read in data
2)      Specify filtering criteria (current : Point2D)
3)      Specify sorting criteria (current : x-ordinate)
4)      Specify sorting order (current : ASC)
5)      View data
6)      Store data

Please enter your choice : 1

Please enter filename : messy.txt

13 records read in successfully!

Going back to main menu …
```

Impt !! Please include your:
1. Student ID
2. Student Name

every time you display your Main Menu. Marks will be deducted if the required info is not shown.

The figure on the left describes a sample interaction for specifying input filename to read in data.

The program should acknowledge by indicating the no. of records read in successfully!

```
Student ID      : 1234567
Student Name    : Tan Ah Meng Elvis
-------------------------------------------------
Welcome to Assn3 program!

1)      Read in data
2)      Specify filtering criteria (current : Point2D)
3)      Specify sorting criteria (current : x-ordinate)
4)      Specify sorting order (current : ASC)
5)      View data
6)      Store data

Please enter your choice : 2

[ Specifying filtering criteria (current : Point2D) ]

    a)      Point2D records
    b)      Point3D records
    c)      Line2D records
    d)      Line3D records

    Please enter your criteria (a – d) : d
    Filter criteria successfully set to 'Line3D'!
. . .

1)      Read in data
2)      Specify filtering criteria (current : Line3D)
3)      Specify sorting criteria (current : Pt. 1)
4)      Specify sorting order (current : ASC)
5)      . . .
```

The figure on the right describes a sample interaction for specifying filtering criteria to indicate which **type of records** user wish to see.

Note1: Observe that the current filtering criteria has changed in option 2) of the main menu, once it has been successfully set to 'Line3D' !

Note2: Observe that the current sorting criteria is **automatically changed** to the default for Line3D records, which is sorting by Pt 1's (x, y) coordinates. (refer to Appendix E for details)

APPENDIX F (con't)

(Sample Menu Interactions)

```
. . .

1)      Read in data
2)      Specify filtering criteria (current : Line3D)
3)      Specify sorting criteria (current : Pt. 1)
4)      Specify sorting order (current : ASC)
5)      View data
6)      Store data

Please enter your choice : 3

[ Specifying sorting criteria (current : Pt. 1) ]

    a)      Pt. 1's (x, y) values
    b)      Pt. 2's (x, y) values
    c)      Length value

    Please enter your criteria (a – c) : c
    Sorting criteria successfully set to 'Length'!
. . .

1)      Read in data
2)      Specify filtering criteria (current : Line3D)
3)      Specify sorting criteria (current : Length)
4)      Specify sorting order (current : ASC)
5)      . . .
```

The figure on the right describes a sample interaction for specifying sorting criteria to indicate which **attribute, of the selected type of records** user wish to see.

Note3: Observe that the sub-menu display options which are relevant to currently selected filter, which is 'Line3D'! (refer to Appendix E for details)

Note4: Observe that the current sorting criteria has changed in option 3) of the main menu, once it has been successfully set to 'Length'!

```
. . .

1)      Read in data
2)      Specify filtering criteria (current : Line3D)
3)      Specify sorting criteria (current : Pt. 1)
4)      Specify sorting order (current : ASC)
5)      View data
6)      Store data

Please enter your choice : 4

[ Specifying sorting order (current : ASC) ]

    a)      ASC (Ascending order)
    b)      DSC (Descending order)

    Please enter your criteria (a – b) : b
    Sorting order successfully set to 'DSC'!
. . .

1)      Read in data
2)      Specify filtering criteria (current : Line3D)
3)      Specify sorting criteria (current : Length)
4)      Specify sorting order (current : DSC)
5)      . . .
```

The figure on the right describes a sample interaction for specifying sorting order to indicate whether records are to be displayed / output in Ascending or Descending order.

Note5: Observe that the current sorting order has changed in option 4) of the main menu, once it has been successfully set to 'DSC'!

APPENDIX F (con't)

(Sample Menu Interactions)

```
. . .

1)      Read in data
2)      Specify filtering criteria (current : Line3D)
3)      Specify sorting criteria (current : Length)
4)      Specify sorting order (current : DSC)
5)      View data
6)      Store data

Please enter your choice : 5

[ View data … ]
 filtering criteria : Line3D
 sorting criteria : Length
 sorting order : DSC

  P1-X   P1-Y   P1-Z      P2-X   P2-Y   P2-Z     Length
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[ 999,   999,   999]    [-999, -999, -999]    3460.638
[-999, -999, -999]    [    3,     3,     3]    1735.515
[-999, -999, -999]    [   -9,    -9,    -9]    1714.730
[ -99,   -99,   -99]    [-999, -999, -999]    1558.846
[    3,     3,     3]    [ 999,   999,   999]    1275.123
[   -9,    -9,    -9]    [ -99,   -99,   -99]    155.885

Press any key to go back to main menu …
```

The figure on the right describes a sample interaction to display data.

Note6: Observe that all the latest criteria specified in main menu options 2) – 4) are re-iterated before the rows of Line3D records are displayed.

```
. . .

1)      Read in data
2)      Specify filtering criteria (current : Line3D)
3)      Specify sorting criteria (current : Length)
4)      Specify sorting order (current : DSC)
5)      View data
6)      Store data

Please enter your choice : 6

Please enter filename : Line3D.txt

13 records output successfully!

Going back to main menu …
```

The figure on the right describes a sample interaction to write data to an output file.

Note7: The contents and its formatting of the data in the output file 'Line3D.txt' should be similar to that displayed when user choose main menu option 5! (please refer to output of the figure above)

APPENDIX G

Submission Instructions (V. IMPT!!)

1) Deliverables

a) All submissions should be in **softcopy**, unless otherwise instructed

b) For the actual files to be submitted, you typically need to include the following:

- word document report (e.g. ***.doc**), save as MS Word 97-2003 format
- the source file(s), (e.g. ***.h**, ***.o**, or ***.cpp** files)
- the executable file, (using Ubuntu g++ compiler), compile into an executable filename with ***.exe** (e.g. csci251_a3.**exe**)

2) How to package

Compress all your assignment files into a single zip file. Please use the following naming format :

<FT/PT>_<Your Grp>_A3_<Stud. No.>_<Name>.zip

Example : FT_TutGrp3_A3_1234567_JohnDoeAnderson.zip

- **<FT/PT>** Use "**FT**" for **F**ull-**T**ime student, "**PT**" if you are **P**art-**T**ime student
- <Your Grp> refers to your SIM tutorial group (e.g. TutGrp1 / TutGrp2 / TutGrp3 / etc.)
- A3 if you are submitting assignment **3**, A1 if submitting assignment **1** etc.
- **<Stud. No.>** refers to your UOW student number (e.g. 1234567)
- **<Name>** refers to your UOW registered name (e.g. JohnDoeAnderson)

3) Where to submit

Please submit your assignment via Moodle eLearning site.

IMPORTANT NOTE :

- To minimize the chances of encountering **UNFORSEEN SITUATIONS** (mentioned below), please do an EARLY SUBMISSION via Moodle.
- Prior to the relevant assignment deadline, you can upload (and replace) your assignment submissions in Moodle **as many times as you like**
- It is your responsibility to confirm that you have submitted the final (and correct version) of your deliverables to Moodle before deadline
- Any submission uploaded to Moodle after deadline will be considered late

In the event of UNFORSEEN SITUATIONS :

(E.g. 3 hrs prior to deadline, there is **proof of** unforseen events like Moodle site down, unable to upload assignment, undersea internet cable damaged by sea urchins, etc)

Please email your single zip file to your tutor at :

csci251@yahoo.com for ***FULL*** TIME students

csci251@yahoo.com for ***PART*** TIME students

In your email **subject** line, type in the following information :

<FT/PT> <Your Grp> <assignment info> <**student number**> and <**name**>

Example:

To : **tutor's email** (see above)

Subject : **FT TutGrp3 A3 1234567 JohnDoeAnderson**

Note 1 : The timestamp shown on tutor's email Inbox will be used to determine if the assignment is late or not.

Note 2 : After email submission, your mailbox's ***sent folder*** would have a copy (record) of your sent email, please **do not delete** that copy !! It could be used to prove your timely submission, in case the Tutor did not receive your email!

4) When to submit

a) Depending on the time-table, a software demo / testing / presentation for your assignment will be scheduled during the:
- 3rd - 5th lab session for the semester (i.e. lab 3 - 5), for Full Time (**FT**) students
- 2nd - 4th lab session for the semester (i.e. lab 2 - 4), for Part Time (**PT**) students

Please consult your tutor for further details. Some time will be allocated for students to demo / present / explain your system design or run test cases during the session.

b) Please refer to the following table on the different deliverables, submission events & deadlines

Assignment #	Submission Deadline (check Moodle for EXACT date-time)		Software Demo / Testing (test cases), during ...	Email Test Case Result files by :
	PT	FT		
1	Lab 2	Lab 3	Lab 2(PT), Lab 3(FT)	End of Lab 2(PT), Lab 3(FT)
2	Lab 3	Lab 4	Lab 3(PT), Lab 4(FT)	End of Lab 3(PT), Lab 4(FT)
3	Lab 4	Lab 5	Lab 4(PT), Lab 5(FT)	End of Lab 4(PT), Lab 5(FT)

Note: (PT) = Part Time Students, (FT) = Full Time Students !

c) IMPORTANT NOTE : Non-submission of any of the above mentioned deliverables will result in ZERO marks! Please check with your Tutor personally if you are unsure!

5) Please help by paying attention to the following …

! VERY IMPORTANT !

PLEASE FOLLOW ALL THE GUIDELINES / REQUIREMENTS IN ALL ASSIGNMENT APPENDICES !!

PLEASE FOLLOW ALL THE SUBMISSION INSTRUCTIONS FROM **1** TO **4** !!

IF YOU ARE **NOT SURE**,

PLEASE **CHECK WITH YOUR TUTOR** DURING LABS / LECTURES !

OR ...

PLEASE EMAIL YOUR ENQUIRIES TO YOUR TUTOR !

MARKS WILL BE DEDUCTED IF YOU FAIL TO FOLLOW INSTRUCTIONS !!

Examples of marks deduction :

- Your deliverables / zip file does not follow naming convention
- You have no email subject / or do not following naming convention
- Your email address / content does not include your name/identity (i.e. tutor cannot easily identify sender)

- Wrong naming or **misleading information** given
  (e.g. putting “A2” in email subject, when you are submitting “A1”)
  (e.g. naming “A1” in your zip file, but inside contains A2 files)

- You email to the WRONG tutor
- Your submission cannot be downloaded and unzipped
- Your program cannot be re-compiled and/or executable file cannot run
- Your report / testing files cannot be opened by Microsoft Word / Excel 2003
- You did not submit / incomplete submission of software demo / testing files
- etc

6) Re-submission administration

After the deadline, (on case-by-case basis), some students / grp may be granted an opportunity for an un-official resubmission by the tutor. If this is so, please adhere to the following instructions carefully:

<Step 1> - Prepare 2 zip files as follows :

Zip up for re-submission, program files according to the following format :

Resubmit_<FT/PT>>_<Your Grp>_A3_<Stud. No.> _<Name>.zip

Example : **Resubmit_FT_TutGrp3_ A3_1234567_JohnDoeAnderson.zip**

Zip up for re-submission, test case results files according to the following format :

Resubmit_<FT/PT>_<Your Grp>_A3_TCResults_<Stud. No.> _<Name>.zip

Example : **Resubmit_FT_TutGrp3_A3_TCResults_1234567_JohnDoeAnderson.zip**

- **<FT/PT>** Use "**FT**" for **F**ull-**T**ime student, "**PT**" if you are **P**art-**T**ime student
- <Your Grp> refers to your SIM tutorial group (e.g. TutGrp1 / TutGrp2 / TutGrp3 / etc.)
- A3 if you are submitting assignment 3, A2 if submitting assignment 2 etc.
- **<Stud. No.>** refers to your UOW student number (e.g. **1234567**)
- **<Name>** refers to your UOW registered name (e.g. **JohnDoeAnderson**)
- V. IMPT - To prevent Tutor's Inbox from blowing up in his face, each student is only allowed to re-submit ONCE, for each assignment only!

<Step 2>

Please email your 2 zip files to your tutor's email (refer to section 3) - Where to submit)

In your email **subject** line, type in the following information :

Resubmit <FT/PT> <Your Grp> <assignment info> <**student number**> and <**name**>

Example:

To : **tutor's email** (refer to section 3) - Where to submit)

Subject : **Resubmit FT** TutGrp3 A3 **1234567 JohnDoeAnderson**

IMPORTANT NOTE :

Non-submission of any of the above mentioned files, or failure to adhere to submission instructions will result in ZERO marks!

Please check with your Tutor personally if you are unsure!