OOP3200 – Object Oriented Programming II

C++ Lab 5
Standard Template Library (STL)

Due: Week 7 (Sunday October 25, 2020) @ midnight

Value 6%

STL **Maximum Mark: 100**

Overview In this lab you will implement a **`std::map`** to store data read from a **text file** for processing. The data itself is a number of labelled (or 'keyed') **Vector2D points**, e.g. the first point is (0, 0) and is labelled "**AA**". Subsequent points are also labelled with a two-letter uppercase 'key'.

Your **first task** is to read all the labels and points from the **data file**, then report to the user **how many points there are**, as well as the total distance between all of the points in order (i.e. the distance between AA and AB + the distance between AB and AC + the distance between AC and AD, etc.).

Your **second task** is prompt the user to enter a label (key) and, if the key is in the map, **report the distance** between that point and the start point, i.e. AA (0, 0). You will continue to allow the user to enter in labels until they want to end.

The following files are provided:

- ***Vector2D.h*** – this header file declares and defines the **Vector2D** class. Note that a **Vector2D** object does not store a label, only x and y values. The class features overloaded friend operators **>>** and **<<** for inputting and outputting **Vector2D objects**. You should not need to modify this class.
- ***PointData.dat*** – this is the file that contains the labels and **Vector2D points** that the final version of your program will read into the map and process.
- ***MockDataForTesting.txt*** – this file contains some labels and **Vector2D points** that you can use for testing/debugging your program. The difference between the data in this file and **PointsData.dat** is that the labels/points are in order and are more easily calculated manually. The final version of your program should not read from this file.

Instructions:
Section 1: Requirements for StandardDeck Class
(70 Marks: Functionality)

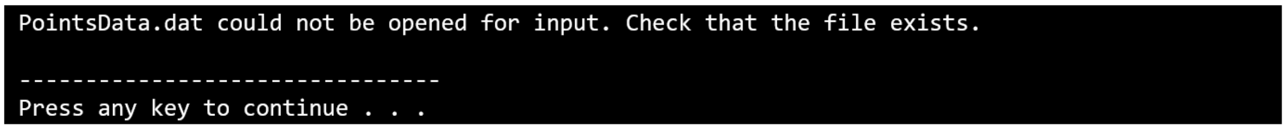
1. **Reading Labels/Points into the Map:** Attempt to open the data file. If the data file opens, read a label, followed by a point and **insert** them into the **map** using the label as the key. Repeat until all the data is read and close the file. If the data file did not open, tell the user and remind them to check that the file exists. If the file opened but the map is empty after the input loop, report that to

the user and remind them to check that the file contains valid data in the correct format. Only continue processing if the file was opened and the map is not empty. (20 Marks: Functionality)

2. **Determine the Total Distance Between All Points in Order:** Use an iterator and a loop to traverse each label/point in the map. For each label/point, determine the distance from that point to the previous point (or next point depending on how you implement this) and add that distance to a total. Note that the **Vector2D** class includes a static distance function that can be used to determine the distance between two **Vector2D** objects, so you should not need to use any complicated math here. Report to the user how many points the map contains and what the total distance is (20 Marks: Functionality).
3. **Determine the Distance Between the Start Point and a User Selected Point:** Prompt the user to enter a label or to enter "quit" to end. If the user entered anything other than "quit", attempt to find the label they entered in the map. If it was found, report the distance between the point for the label they entered and the start point (i.e. the point labelled "AA"). Otherwise, tell the user that the label they entered is not in the map. Repeat these steps until the user enters "quit". (20 Marks: Functionality).
4. **Exception Handling.** Catch any **`std::exception`** thrown. Report to the user that a run-time error occurred and show what exception was thrown. (10 Marks: Functionality).

Example Outputs:

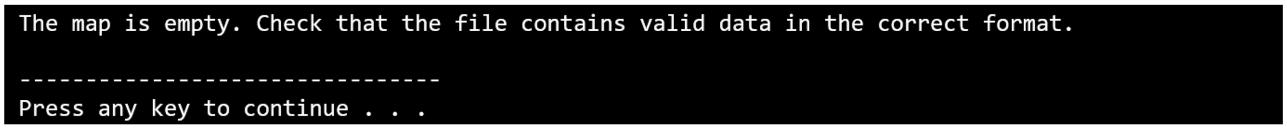
The data file did not open (file name misspelled):



```
PointsData.dat could not be opened for input. Check that the file exists.

--------------------------------
Press any key to continue . . .
```

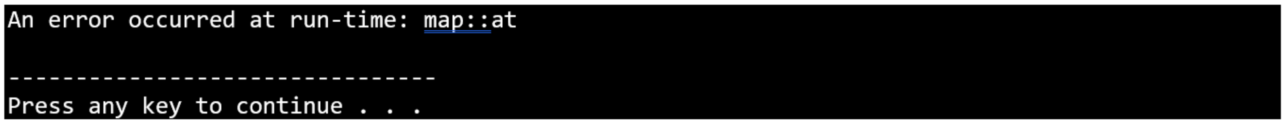
The data file opened but nothing was read into the map successfully (read from an empty file):



```
The map is empty. Check that the file contains valid data in the correct format.

--------------------------------
Press any key to continue . . .
```

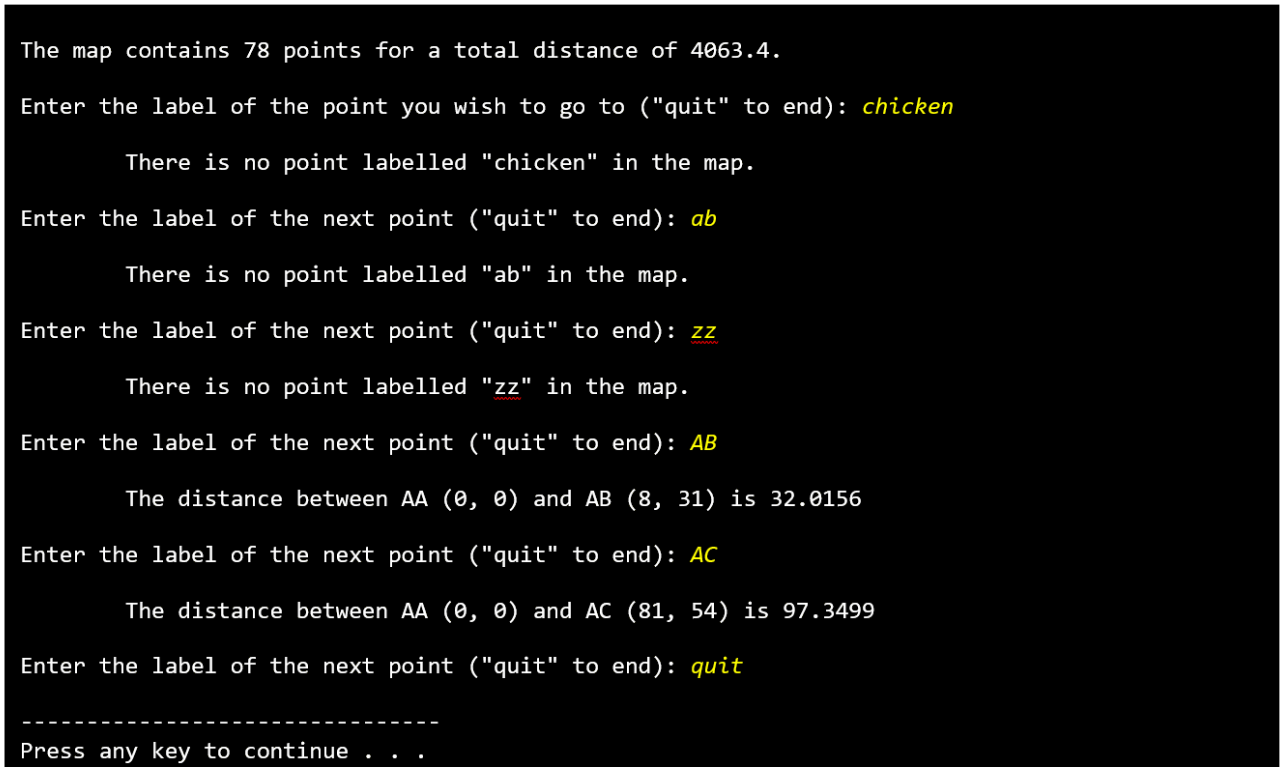
An exception gets thrown (example, not likely to happen):



```
An error occurred at run-time: map::at

--------------------------------
Press any key to continue . . .
```

The map was successfully filled, various labels attempted (user input is shown in yellow):



```
The map contains 78 points for a total distance of 4063.4.

Enter the label of the point you wish to go to ("quit" to end): chicken

        There is no point labelled "chicken" in the map.

Enter the label of the next point ("quit" to end): ab

        There is no point labelled "ab" in the map.

Enter the label of the next point ("quit" to end): zz

        There is no point labelled "zz" in the map.

Enter the label of the next point ("quit" to end): AB

        The distance between AA (0, 0) and AB (8, 31) is 32.0156

Enter the label of the next point ("quit" to end): AC

        The distance between AA (0, 0) and AC (81, 54) is 97.3499

Enter the label of the next point ("quit" to end): quit

--------------------------------
Press any key to continue . . .
```

Section 2: Program Requirements

(10 Marks: Functionality)

1. Your `main()` function should implement all the features detailed above. How you do this is up to you (10 Marks: Functionality).

Section 3: General Requirements

(5 Marks: Internal Documentation, 5 Marks: Version Control, 10 Marks: Demo Video)

1. Include **Internal Documentation** for your site **(5 Marks: Internal Documentation):**
    a. Ensure you include a **comment header** for your **C++ Files** that indicate: the **File name**, **Student's Name**, **StudentID, and Date,** (2 Marks: Internal Documentation).
    b. Ensure you include **method and function headers** for all of your **class methods,** and any **utility functions** (1 Marks: Internal Documentation)
    c. Ensure all your code uses **contextual variable names** that help make the files human-readable and follow the C++ style guide (1 Marks: Internal Documentation).
    d. Ensure you include **inline comments** that describe your program's functionality where appropriate. **Note:** Please avoid "over-commenting" (1 Marks: Internal Documentation)
2. Share your files on **GitHub** to demonstrate Version Control Best Practices **(5 Marks: Version Control).**
    a. Create an appropriately named GitHub Repository that you and your partner will use for this lab. Only one repository is required (1 Mark: Version Control)
    b. Your repository must include **your code** and be well structured (2 Marks: Version Control).

   c. Your repository must include **commits** that demonstrate the project being updated at different stages of development – each time a major change is implemented (2 Marks: Version Control).
3. Create a Short Video presentation on **YouTube** or another streaming provider. You must include a short **PowerPoint** (or Google Slides) Slide Deck that includes a **single slide** to start your video (10 Marks: Video)
   a. The **first** (and only) **Slide** of your Slide Deck must include **current images** of you and your partner (no avatars allowed) that are displayed appropriately on the page. You must also include your **Full Names**, **Student IDs**, the **Course Code**, **Course Name,** and your **Assignment** information. (2 Marks: video)
   b. You or your partner will **demonstrate** your program's functionality. You must show your program working properly in the console (2 Marks: Video)
   c. You or your partner will **describe** the code in your files that drives the functionality of your program (2 Marks Video).
   d. Sound for your Video must at an appropriate level so that your voices may be **clearly heard,** and your screen resolution should be set so that your program's code and console details are **clearly visible** (2 Marks: Video).
   e. Your Short Video should run no more than 5 minutes (2 Marks: Video).

SUBMITTING YOUR WORK

Your submission should include:

1. A zip archive of your program's Project files uploaded to DCConnect.
2. A working link to your appropriately named GitHub repository
3. A working link to your demo video posted on YouTube or another streaming provider

Evaluation Criteria

Feature	**Description**	**Marks**
Functionality	Program deliverables are me and site functions are met. No errors, including submission of user inputs.	80
Internal Documentation	File headers present, including file & student names & description. Functions and classes include headers describing functionality & scope. Inline comments and descriptive variable names included and follow style guide.	5
Version Control	GitHub repo created with commit history demonstrating regular updates. 2 marks for initial commit. 2 marks for working with GitHub throughout the development process	5
Video Presentation	Your short video must demonstrate your site and describe your code. Your audio must be at an appropriate level and your screen must be clearly seen.	10
Total		**100**

This assignment is weighted **6%** of your total mark for this course.

Late submissions:

- 20% deducted for each day late.

External code (e.g. from the internet or other sources) can be used for student submissions within the following parameters:

1. The code source (i.e. where you got the code and who wrote it) must be cited in your internal documentation.
2. It encompasses a maximum of 10% of your code (any more will be considered cheating).
3. You must understand any code you use and include documentation (comments) around the code that explains its function.
4. You must get written approval from me via email.