

assignment 2 documentation

ICT283 Data Structures and Abstractions



Thien Twee Zian

34664971

Table of Contents

[Rationale for Design 3](#_Toc141731215)

[Application Test Plan 5](#_Toc141731216)

[Test Plan Notes 5](#_Toc141731217)

[Test Plan 6](#_Toc141731218)

[Test Plan Screenshots 6](#_Toc141731219)

[Test 1 6](#_Toc141731220)

[Test 2 6](#_Toc141731221)

[Test 3 6](#_Toc141731222)

[Test 4 6](#_Toc141731223)

[Test 5 6](#_Toc141731224)

[Test 6 6](#_Toc141731225)

[Test 7 6](#_Toc141731226)

[Test 8 6](#_Toc141731227)

[Test 9 6](#_Toc141731228)

[Test 10 6](#_Toc141731229)

[Test 11 6](#_Toc141731230)

[Test 12 7](#_Toc141731231)

# Rationale for Design

* Individual classes to store singular attribute of the wind record which requires methods and additional information storage (Date class for date, Time class for time etc)
* Compilation of wind record attributes in WindData class, with objects from respective classes, as well as for wind speed, air temperature and solar radiation.
* WindDataStore to store the individual records of wind data as a collection class. This class uses maps and the template BST to store the WindData.

In my data structure design for WindDataStore, I have used the STL map to store separate records of BSTs containing the WindData. The map using the month and year of the record as a string key to separate the BST according to this filter. This makes every instance of a map record storing only data pertaining to that particular month and year.

The BST uses the date (as a whole, inclusive of day, month, year) and the time as a key to ensure the records kept are unique. From this, the BST would store individual WindData objects into its nodes, with every BST created representing a particular month and year.

The rationale of doing so is to reduce the height of the BST, as the time complexity for operations for perfectly balanced BST is O(log n), where n is the number of nodes in the tree. However, in a unbalanced BST, it will become a linked list, which has a time complexity of O(n), where n is the number of nodes in the list.

To mitigate the problems of the BST storing too many nodes, thus affecting the time it takes to run operations, I used a map to segregate the data to be stored by the month and year. By doing so, we will have more than one BST in the overall structure, however the number of nodes to be stored would be reduced, thus improving the time complexity of BST operations.

Maps are chosen as it accepts keys by default (STL map). This reduces the need to create a method of appointing unique keys. The STL map is also sorted by default, thus improving operations done with it.

The STL Vector is also used in this program, for certain attributes such as m\_headerNames in the WindDateStore class. The reason for doing so is because vectors have the ability to allow adjustable storage space. For the m\_headerNames, as currently only certain attributes of the wind data are stored, only those header names are stored. However, if in the future other information are required to be retrieved, new header names would be required. Thus, the definite size for the array cannot be determined, and vectors are used to future proof the design, reducing the amount of work required for future coding if more information is required to be stored.

The node class is designed using structs instead of classes, as there are no methods required to be in the struct, and the nodes are purely used only with the template BST, it is not supposed to and should never be retrieved out of the template BST class.

**Conclusion**

Even though using the combination of the template BST and STL map would improve the time complexity of operations, this would increase the time taken for initial storage of data from file to system, as more than one BST are created, and additional runtime is taken to ensure that the BST trees are stored in the right map nodes.

However, after storage of the data is completed, the program will see a significant improvement in running operations, as opposed to,

* One BST containing everything, where during operations, a very high number of nodes have to be traversed in order to complete said operation.
* One map containing everything, which would experience similar problems with using one BST, even if it is perfectly balanced.

In conclusion, by “mapping” the BSTs by month and year, we should see an significant improvement in operation runtimes.

# Application Test Plan

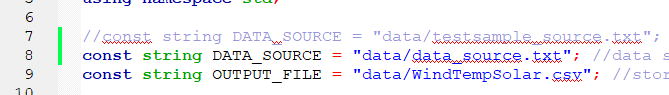
## Test Plan Notes

The application test plan is run by using a separate test file called testsample\_source.txt, stored in the data folder. Within the txt file has the names of the csv files the application is to load from, which are also located in the data folder.

The purpose of this separate test data is to speed up testing, as using the original data source file takes a long time to load, due to the large amount of data, and the data structure involved. This problem is even more visible when running Menu option 5, which deletes all information stored in the data structures within the system, and reloads them again according to the data source.

Therefore, to ease testing, the sample data files are created. The samples are created with the application in mind, with data spread across multiple files, and with both varying and similar months and year (for option 3).

When submitted for marking, the default data source will be as per given to us (data/data\_source.txt). I have left the code for utilising the test sample files above the default data source (lines 8 and 9). To use the sample data, simply comment out the current default at line 9 and uncomment the code on line 8. The original data provided has also been tested and is working.

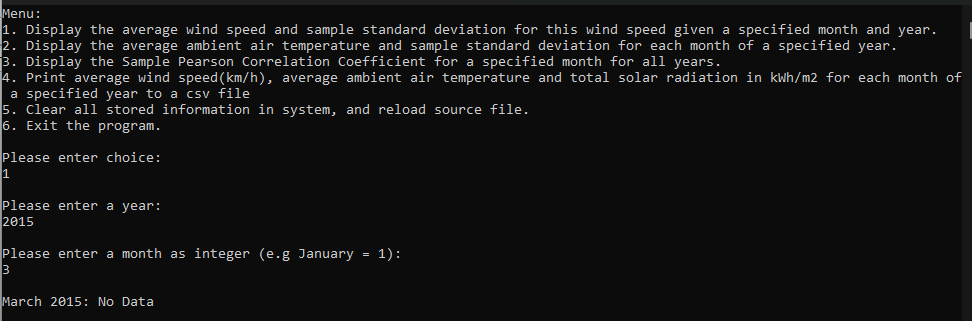


## Test Plan

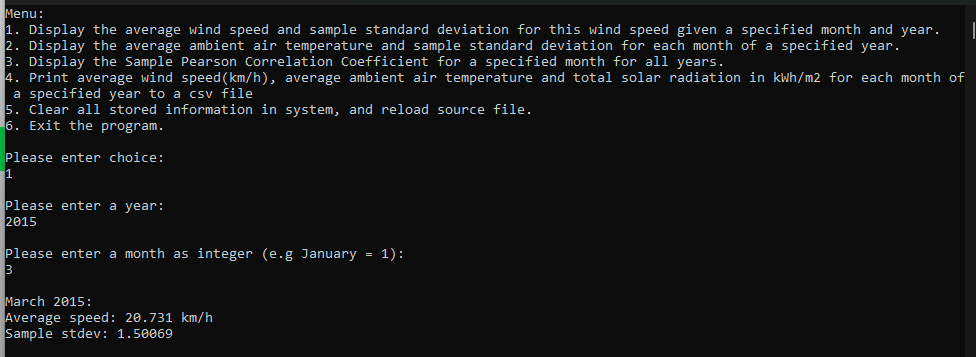


# Test Plan Screenshots

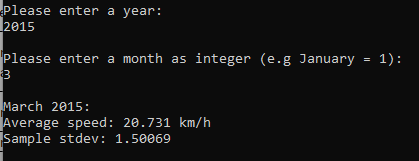
## Test 1



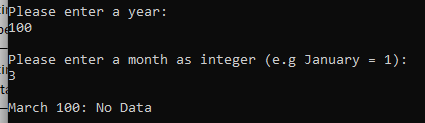
## Test 2



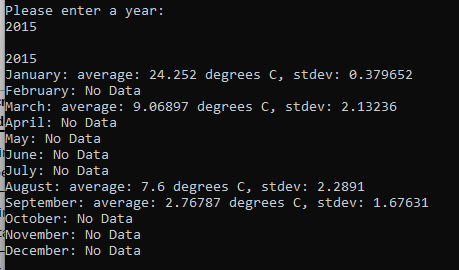
## Test 3



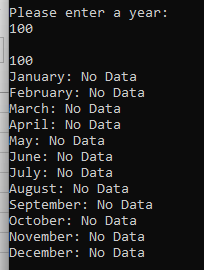
## Test 4



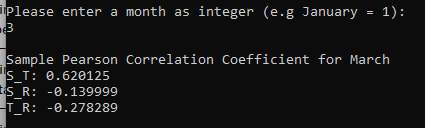
## Test 5



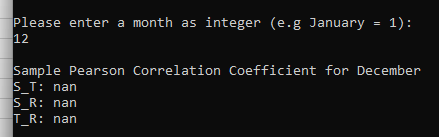
## Test 6



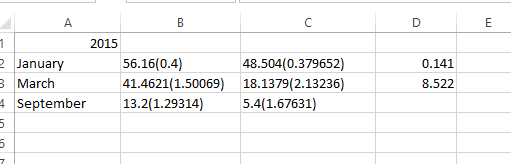
## Test 7



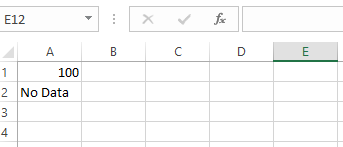
## Test 8



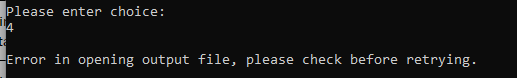
## Test 9



## Test 10



## Test 11



## Test 12

