Rationale for Data Structures

When deciding how data structures are to be used in the program (in particular, the template BST and STL map), my immediate goals were to keep the program design as simple as possible without adding unnecessary complexity and to have a minimal loading time during the program's execution. The issue of duplication, sorting and searching of data was something that needed careful thought to achieve these goals.

READING FROM FILE AND STORING DATA – STL Map

When reading in each line from the file, I ended up deciding to store the data inside an STL map data structure. However, I needed a way to compare each line against one another to enable the ordered storing of data to occur as well as KEY and VALUE types. My original WeatherType struct (in Assignment 1) contained a Date and Time object, as well as the other measurements. I decided to 'break up' the struct into two parts. The first (called 'DateTime') contains a Date and Time object and will act as the KEY of the map data structure. Equals and less than operators were provided to enable the DateTime struct to be used as the KEY. The second struct ('WeatherData') contains the speed, solar radiation and ambient air temperature and will act as the VALUE of the struct. The original WeatherType struct will remain and will instead contain DateTime and WeatherData type variables. The data stored in the map will be in date and time order (earliest date and time to most recent). Duplicate date and times contained in the files will be ignored at the time of insertion into the map.

The decision to use the STL map in this way was for several reasons. Firstly, the insertion of data occurs (mostly) at logarithmic time O(logN), reducing the loading time of the files into the program. Secondly, the data will be sorted immediately after reading from the files and will be helpful in the later execution of the program (as mentioned later). Lastly, the map data structure allows for dynamic resizing to allow flexibility in the amount of data that can be stored. However, using a hash functionality would have meant a near-constant time O(1) insertion of data as well as searching. This would have benefits of having a quicker loading time compared to using the STL map in the way I ended up deciding on. However, after testing my approach on large amounts of files and getting an average 6 second load time on the data provided, I was satisfied with the result. In addition, as hash functionality means the data is not sorted in any particular order, I decided to keep with my initial approach.

SEARCHING AND DATA RETRIEVAL – BST, Vector and STL Map

In terms of searching the data and how the STL map structure would work in the overall program execution, it was challenging to think of a solution. I initially thought of inserting the STL map data structure as stated above into the templated BST, broken down into each year and further again into individual months. This would have meant a significant rework of Assignment 1 to achieve this. In addition, the hash functionality was considered to solve the balanced ordering of data in the BST to ensure logarithmic search time.

Instead, I decided to maintain the use of Vectors (as with Assignment 1) that will be complemented by the use of the BST data structure. Data from the STL map (used for the initial reading from the files) will be inserted in date and time order back into a Vector of WeatherType (containing the struct variables 'DateTime' and 'WeatherData'). I decided this approach as I no longer needed the map functionality for this data, and the basic array type structure would be sufficient for access to data (as mentioned below). Adding complexity by encapsulating this map data structure into a BST would complicate searching and create the issue of having an unbalanced BST and linear O(n) search time (in a worst-case scenario) without adding additional functionality like a hash function.

The BST will only contain each month (as a word string) as well as starting and ending index values as they appear in the Vector. The month string will be used as the search criteria in the BST, and therefore, only the search functionality of the templated BST will be used in the program. The words will be sorted 'alphabetically' and NOT by 'calendar' order. This means iterating once through the Vector (in linear time) to obtain all the data required. Searching the entire Vector will not be required later. However, this also meant encapsulating this data using a data structure first as the BST can only accept one 'type' of data. The data was stored inside a class called 'PeriodIndex'. A class was decided over a struct as there is special behaviour associated with the variables and any change would cause incorrect calculations and results in the program.

To account for the different years, the BST containing the PeriodIndexes for each month will be inserted into a map as a value. The years will act the keys. This will enable an easy and quick search without having massive amounts of data stored inside the BST or the STL map data structures. These searches will occur at the time the user enters a specified period (month, a year or entire date depending on the menu choice). When the indexes are obtained for the specified period, the data contained within the Vector indexes will contain a months' worth of data that will be inserted into a new Vector (if data exists for the period) and used for calculations exactly the same way as was done in Assignment 1.

The end approach I used means Vectors are still used, but the STL map and BST make for quick efficient retrievals of the data that is required to be calculated on in the program. The additional bonus question (Menu option 5), meant the specific day provided by the user needed to be searched linearly inside the Vector (containing the months' worth of data). I considered creating an additional structure to separate the days but decided against this as testing revealed minimal load time to produce a result. It would have provided next to no benefit to the end user.

Overall, it was challenging to come up with an ideal solution to incorporate the STL map and templated BST data structures. Using the approaches outlined in this document aimed to achieve my initial goals of a simple, understandable program that also has a minimal load time. While not a perfect solution, I believe the program executes well enough and satisfies the requirements of the brief.

END OF RATIONALE