Temperature Queries

Points

Points	
10	Design
175	Working Program
15	Code Review
200	**TOTAL**

Submission

1. **Design**: Submit the PDF to Gradescope.
2. **Source Code**: Submit the source code (**LinkedList.h, LinkedList.cpp, Node.h, TemperatureDatabase.h, TemperatureDatabase.cpp, and main.cpp** files) to Vocareum

Specifications

You will implement a linked list to organize temperature sensor readings from a file. Another file will provide queries to report based on the data.

Design

1. Algorithms for determining the average and mode as specified
2. Test cases

Program

You will implement a program that receives as input two files:

1. **A temperature file (e.g., temps.dat)**

   This file contains historic average temperatures for several locations in Texas. The data has been obtained from the United States Historical Climate Network (USCHN). Each line of the file contains a location, a date, and the average temperature (in Celsius) for that location and date. A string, corresponding to a meteorological station id, represents a location. For example, “411048” corresponds to Brenham, Texas. A date is specified by two integers: a year and a month. A sample line in the temperature file would be:

   ```
   410120 1893 2 6.41
   ```

   The temperature value -99.99 is used by the UCHN agency to represent missing information, for example:

   ```
   410120 1893 1 -99.99
   ```

   Valid temperatures are assumed to be in the interval -50.0 to 50.0 (remember that they use Celsius, so this is a good range). The first valid year is 1800. The latest valid year should be our current year. You can declare a constant in your program to specify the current year. (If we were programming this to be used for real, we would instead obtain the current year from the system, so that no code changes are required for future years.)

2. **A query file (e.g., queries.dat)**

   This file contains a list of queries (i.e., requests for information) that your program should calculate based on the data in file temps.dat. Each line in the file contains one query. There are two types of queries:

a. **AVG:** given a location (specified as a station id) and a range of years (specified as two integers, for example, 2000 2006), it computes the average temperature for the location and period. If no data is available for the location/period, the answer will be the string "unknown".

b. ***MODE:*** *given a location and a range of years, it identifies the most common rounded temperature value in the period.*

   i. If no data is available for the period, the expected result is the string "unknown".

   ii. If more than one mode, you should return the one with the highest value.

   iii. Notice that the temperature values in file temps.dat are double numbers. You should round these values to the nearest integer before computing the mode. For example, 8.03, 8.1, and 8.5 are all rounded to 8; 8.51 and 8.8 are rounded to 9. So with this set of data, there are three eights and two nines. And the mode would be 8.

Your program should read these two files and generate a file result.dat with the queries and their results. Regardless,

- temps.dat might look like this:

```
411048 2015 1 9.58
411048 2015 3 14.82
411048 2016 4 20.51
411048 2016 1 10.99
411000 1973 1 0.54
411048 2016 3 18.40
411048 2016 5 -99.99
```

- queries.dat might look like this:

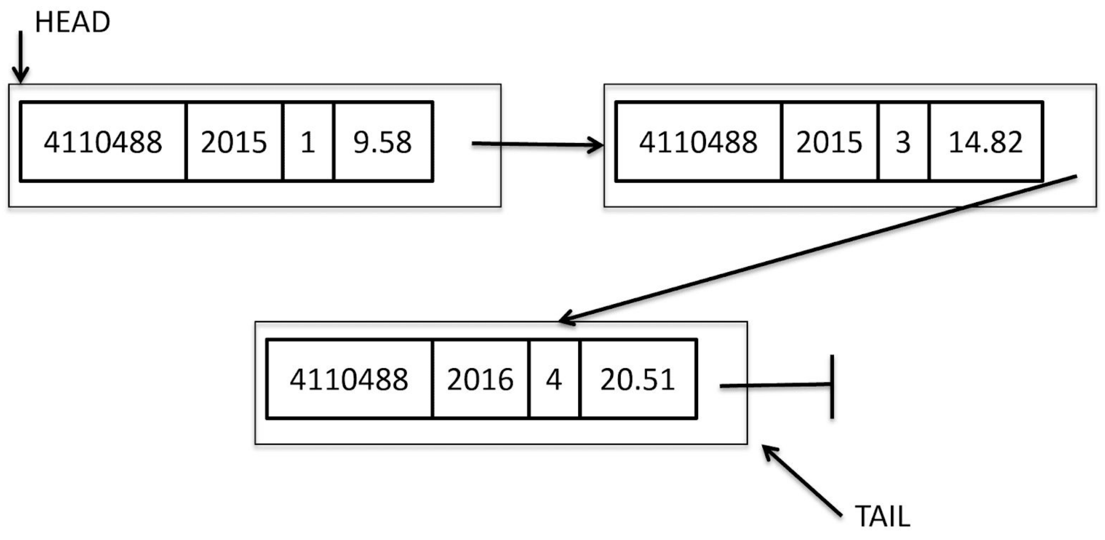
  ```
  411048 AVG 2015 2016
  411138 AVG 1995 1995
  411048 MODE 2015 2016
  ```

Then your program is expected to produce a file named "result.dat" with the following content:

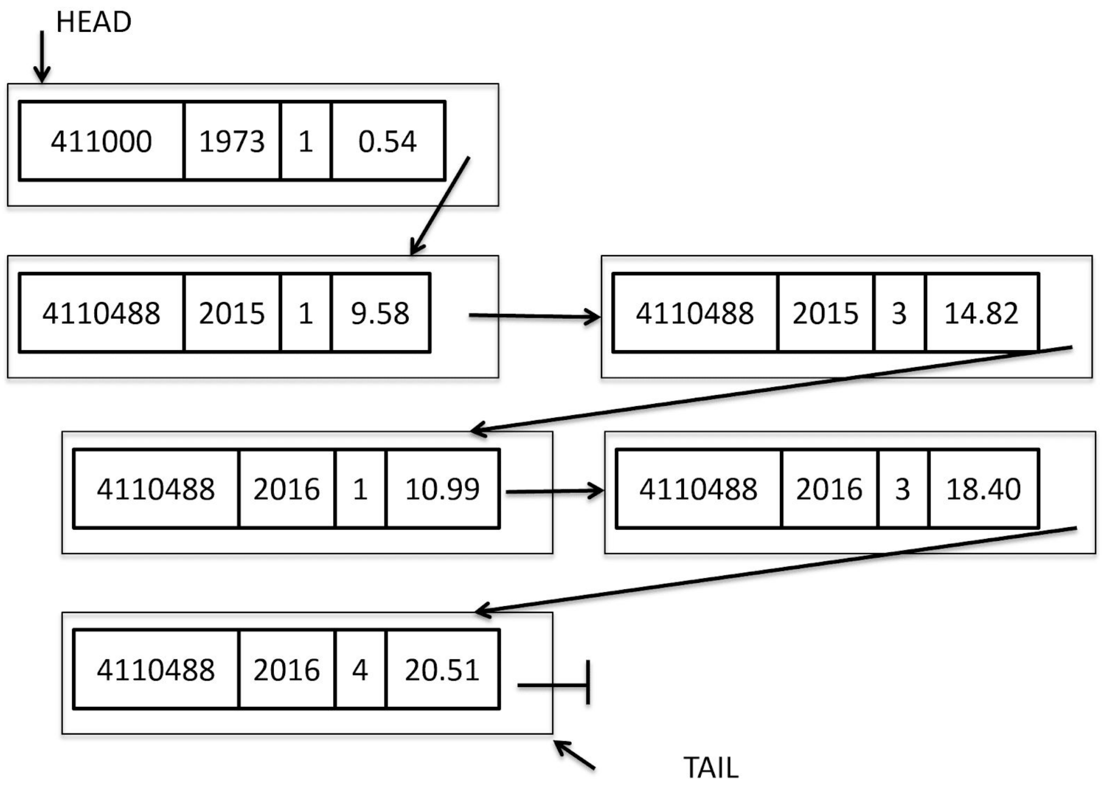
```
411048 2015 2016 AVG 14.86
411138 1995 1995 AVG unknown
411048 2015 2016 MODE 21
```

Requirements

1. You must use this skeleton code to create your program.
2. Your implementation has to use a linked list to store the data you read from the temperature file.
    - You *must* implement a function named "getHead()" in the class LinkedList. This function is supposed to return the pointer to the start Node of the list.
    - You *must* implement an overloaded `operator<` for `struct Node` and you must use this operator to to compare `Node`s while inserting into the `LinkedList`. The nodes in the linked list should:
        - First ordered by location
        - Then by date *(i.e. by year and then by month)*
    - For example, for the sample temps.dat above, after reading 3 lines of the file your linked list looks like:



After reading all 7 lines from the file, it looks like:



Notice that we ignored the entry with value -99.99.

3. Your program should receive the names of the input files as command line arguments. The first argument will be the temperature file, the second the queries file.
4. The output file created by your program should be named **result.dat**.
5. If the input files contain a line with an invalid format, you should output on the console an error message and finish executing the program.
    - Do not throw an exception.
    - Output the message in the console beginning “Error: “ followed by a short description of the error. We do not care what the description looks like. Some examples
        - Error: Invalid temperature
        - Error: Invalid year
    - You can be more specific about the error if you want such as including the invalid value since it may be useful for your own debugging, in case you are making a mistake when validating the input data.

    - Examples of lines with invalid format for temps.dat:

      ```
      4111000 1889 0 -4.3

      4111000 1889 18 -4.3

      4111000 2016 01 -1222.11

      4111000 1889 8.3
      ```

- Examples of lines with invalid format for queries.dat:

  `AVG 2015 2016`

  `411138 AVG 1996 1995`

  `411048 MODE 1500 2016`

  `411048 MAX 2015 2016`

6. You are required to use classes and comply with the “Rule of 3” (see zyBook and slides). More specifically, you need to implement the constructor, destructor, copy assignment, and copy constructor.
   - The constructor for your class representing temperature values should take all data items (station id, year, month, average temperature value) as parameters. Use initialization in the “membername(value)” format (see zyBook and slides)
7. Implement the overloaded output operator for the linked list, such that it inserts the contents of each node to an ostream, separated by a space, and terminated by a line feed:

```
411048 2015 1 9.58
411048 2015 3 14.82
411048 2016 4 20.51
```

Sample Input/Output Files

The following zip file has sample data, query, and result files: tempQueryFiles.zip

- Data: temp-3lines.dat, temp-7lines.dat, temp-70s.dat, temp-2000s.dat
- Queries: queries.dat, queries-70s.dat, queries-2000s.dat
- Results: result-3lines.dat, result-7lines.dat, etc.

Hints & Comments

- You may want to print out your linked list so that you can make sure that you are maintaining it in order as required. First test with the small sample input files that we provide (e.g., first the one with 3 lines, then the one with 7 lines of data.), output your linked list, and inspect it to see if it is representing the input data correctly.
- When you round a double value to calculate the mode, make sure that you are not just truncating the value: the rounded value for 10.6 should be 11, not 10.
- It is often a good idea to develop your solution incrementally, completing and testing components of your overall program. For example, you may want to program and test the part where you read the temperature input file first, before starting to write the code to read the query file. This will help you to avoid making the same mistake in different parts of your code.
- Notice that, as specified, the program you wrote is not very useful, as we often are not interested in average over years. We would prefer to be able to compare averages of specific months across year ranges. Once you complete your program, think about what you would have to do to be able to provide averages for specific months.
- If you tried your solution with very large input files, you may have noticed that your solution takes some time to provide the answers. Students in the Computer science and Computer Engineering majors have the opportunity to take many other courses (data structures, databases, computer systems, operating systems, data mining, distributed systems,

parallel computing) where they learn concepts and techniques to achieve efficient queries over large datasets.