HW: Temperature Queries

You are an amateur meteorologist tasked to develop a series of low-powered temperature querying devices for deploying onto the fields of small farming businesses, in order to provide better agricultural analytics at lower expenses.

Due to the computing limitations of these deployed systems that are passively powered and with minimal storage space, and also the amount of times that you must constantly insert and remove data asynchronously from the different devices, you have decided to implement the system using a linked list data structure for its lower memory footprint and faster runtime complexity.

In order to guide the design and development of the querying devices, you initially work with collected temperature data for reference. In this assignment, you **implement** the code that stores, accesses, and modifies temperature data in a linked list data structure.

# Objectives

* Read data into a linked list data structure.
* Practice reading and validating data from an input file.
* Practice using operator overloading to compare data.

# Points

This homework consists of the main program and an extra credit opportunity.

* **100 points:** The program runs correctly.
* **20 points:** The extra credit runs correctly.
* **120 points:** The maximum number of points (full credit is 100).

# Submission

Get the [starter code](https://drive.google.com/drive/folders/15ZaMxErg98YJptZHvzS2VFEkHcTu9HB2?usp=sharing) and submit the following:

* **LinkedList.cpp** and **LinkedList.h**
* **Node.cpp** and **Node.h**
* **TemperatureData.cpp** and **TemperatureData.h**
* **TemperatureDatabase.cpp** and **TemperatureDatabase.h**
* **main.cpp**

# Specifications

* **Part 1 (Main Program):** You will implement a linked list to organize temperature sensor readings from a file.
* **Part 2 (Extra Credit):** Process a file with queries to report based on the data.
  + **Note:** Autograder support for Part 2 will be released this weekend.

# Program

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

## 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” (string not number) 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.)   
  
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

## Extra Credit: 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:

1. **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”.
2. **MODE:** given a location and a range of years, it identifies the most common rounded temperature value in the period.
   * If no data is available for the period, the expected result is the string “unknown”.
   * If there is more than one mode, you should return the one with the highest value.
   * 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

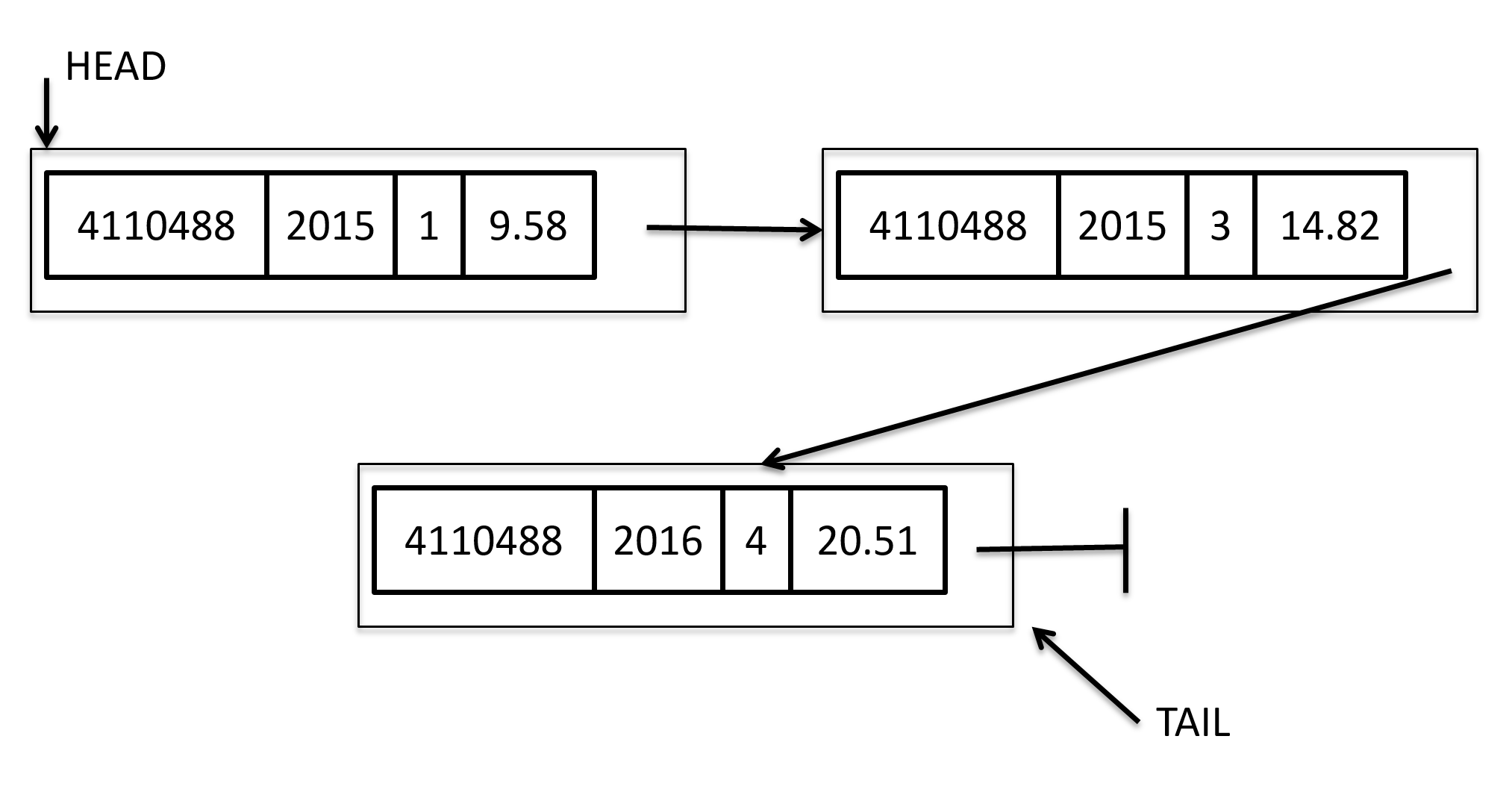
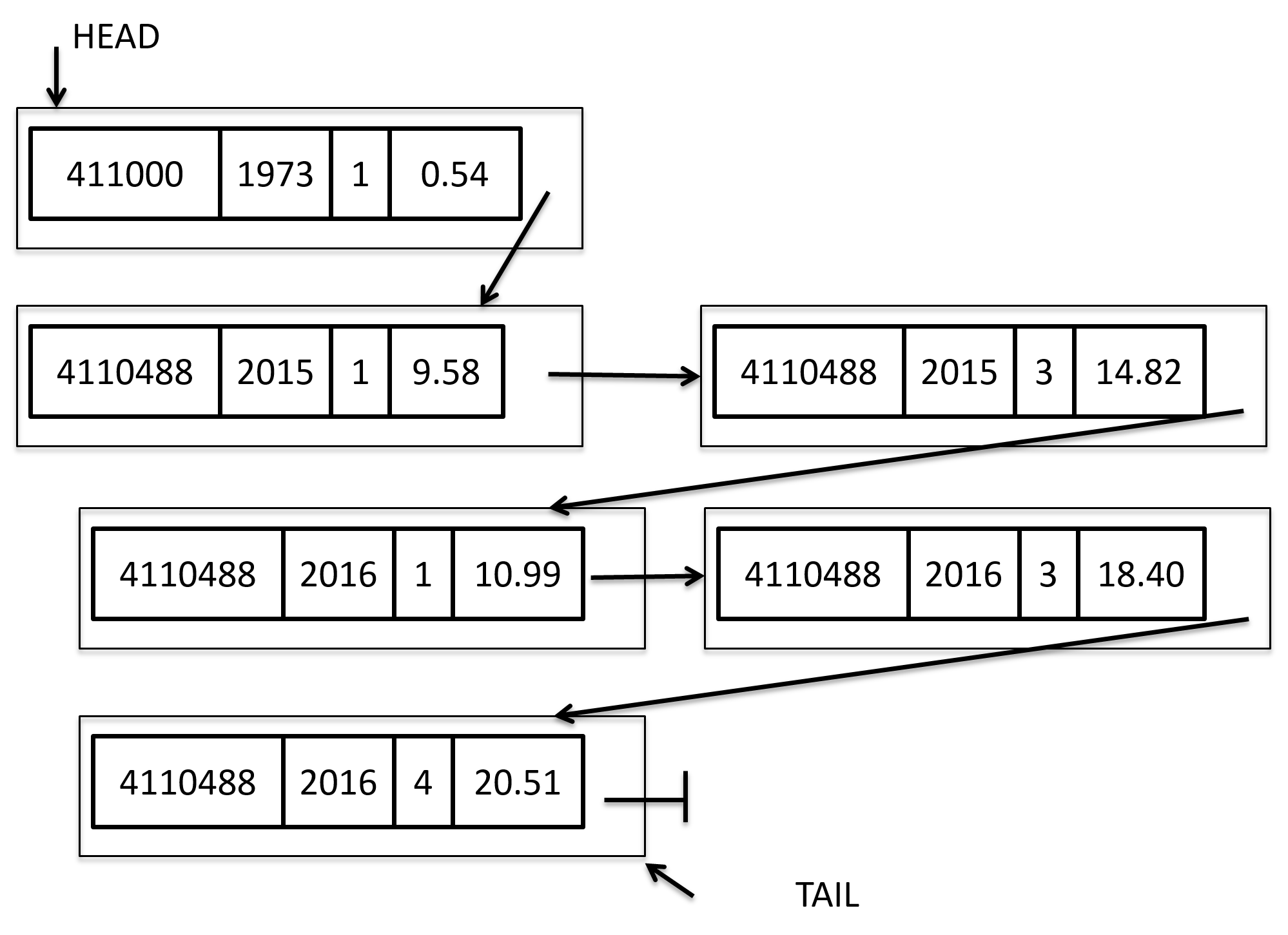
Then your program is expected to produce a file named **result.dat** with the following content:

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

# 

# Requirements

## Part 1

* Download this skeleton code on Gradescope to create your program.
* Approved includes
  + iostream
  + sstream
  + string
  + fstream
  + limits
  + cstdlib
  + vector
  + ctime
  + LinkedList.h
  + Node.h
  + TemperatureData.h
  + TemperatureDatabase.h
* 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 should **not** use this function in any of your code since it is only there to assist autograding.
  + You *must* implement an overloaded operator< for the struct Node (and TemperatureData it calls) and you must use this operator to compare Nodes while inserting into the LinkedList. The nodes in the linked list are:
    - First ordered by location
    - Then by date *(i.e. by year and then by month)*
    - You do not have to worry about getting a duplicate entry for a month and year for a specific location.
  + For example, for the prior sample temps.dat, 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.**
* 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.
* If the input files contain a line with an invalid format, you should output an error message on the console (see examples below) and continue executing the program.
  + ***Do not*** throw an exception.
  + def
  + 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 8.3  
    4111000 2015 11
* 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 for the LinkedList class and any other class that uses the heap/freestore, but I don’t think any other class should use the heap.
  + The constructor for your Node and TemperatureData classes 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)
  + We’ll test these directly.
* Implement the print() function for the linked list. Use this function and/or the overloaded output operator to test your results. We’ll test the linked list functions directly for part 1. The expectation is that the list will be printed out in the correct order based on the comparison rules listed earlier.
  + Output format is the same as the format in the input file. Id, year, month, and temperature separated by a space.
  + When you output, it will look like the input file, with each item on one line. However, the order should be sorted according to the requirements above.
  + Formatting of floating point numbers will use what the ostream output operator (<<) does by default. Note that the formatting created by atoi() frequently does not match what the output operator does.

## Part 2 - Extra Credit Requirements

* 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.
* 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 the description shown below followed by the invalid value. For example:
    - Data File
      * Error: Invalid temperature -1221.11
      * Error: Invalid year 2023
      * Error: Invalid month 0
      * Error: Unable to open input.dat
      * Error: Other invalid input
    - Query File
      * Error: Invalid range 1996-1995
      * Error: Unsupported query MAX
      * Error: Other invalid query
      * *Invalid years will be part of an invalid range.*
        + Error: Invalid range 1776-2025
  + Examples of lines with invalid format for queries.dat:  
    AVG 2015 2016  
    411138 AVG 1996 1995  
    411048 MODE 1500 2016  
    411048 MAX 2015 2016
* The output file created by your program should be named **result.dat**.

# 

# Grading

## Part 1

1. We will unit test the LinkedList class, including insert, print, constructors and rule of 3, and overloaded output operator (which is already done for you if you get print to work).
2. We will unit test Node and TemperatureData classes.
3. We will unit test the TemperatureDatabase's loadData() function. We will look for error messages.

## Part 2 - Extra Credit

1. We will unit test the TemperatureDatabase's performQuery() function. We will look for error messages.
2. We will test the entire program, look for error messages, and examine your result.dat file.

# Sample Input/Output Files

* Get Input files to build list on Gradescope
  + temp-3lines.dat, temp-7lines.dat, etc.
* Extra Credit
  + Get Query Files on Gradescope
  + Get Result Files on Gradescope

# Hints & Comments

## Part 1

* Write a myTest() function for the TemperatureDatabase. You can call that function from main after loading the Database to test things like the rule of three, print(), etc.
* Use the print() function 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.
* It is often a good idea to develop your solution incrementally, completing and testing components of your overall program as you develop them.

## Part 2 - Extra Credit

* It is often a good idea to develop your solution incrementally, completing and testing components of your overall program as you develop them.
* 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.
* Think about how you track the different temperatures for determining the mode. Note that you only have to keep track of values from -50 to 50. How many values is that?
* 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, Computing, 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.