**Programming Assignment 2**

**50 points**

Your goal is to perform queries of real weather data. The data we will use is located at <ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/>.

You may wish to develop a sequential program that solves the problem, to debug your sequential code, before you tackle the concurrent solution, but this is not required.

The files ghcnd\_hcn.tar.gz, ghcnd-stations.txt and readme.txt are useful for our assignment. ghcnd\_hcn.tar.gz contains approximately 1,200 files, each one corresponding to a weather station in the United States. Each line in the file contains a weather record for one month.

Your program should run in a particular directory. It should assume that the “ghcnd-stations.txt” file is in that same directory, and that each file in the “ghcnd\_hcn” directory is a file with weather station data. Please do not use any slashes or backslashes in your file handling, as this makes the program difficult to grade in Linux.

The file ghcdn\_stations.txt contains an entry for each file in ghcnd\_hcn.tar.gz, providing geographic data, corresponding to that weather station where those measurements were taken. You should extract the contents of this file to a directory. The compressed version of this file is 274 Megabytes, and the extracted files compose approximately 2.4 Gigabytes, so ensure are using a system that can handle this amount of data.

The readme.txt file documents in detail the format of ghcnd\_\* files.

You should present a set of questions to the user, asking for a starting year, and ending year, starting month, and ending month, and whether to calculate the maximum or minimum temperatures. Your program should print the maximum (or minimum) five temperatures that occurred in that range of years and months.

Your program should use thread pools, futures and callables to complete its processing. First, your program should create a thread pool with a reasonable number of threads, perhaps 4 or 5 times the number of cores on your system. Then, your program should create one future for each file, and execute that future via a callable. Each callable returns data corresponding to your query for one file, as a future. Since it is possible that only one file contains all the results for your query, you should return that many results for each file. Your file must read the following elements of each line: id, year, month, element, all values, all qflags. Your program should discard the value if qflag is anything other than an empty (space) column.

Second, you should consolidate all the query results from all the files into a single list. You should do this as each query completes.

Third, after all the queries have completed, you should generate a second set of four futures. Those futures should iterate over the consolidated list you just constructed, and find the top results for that query.

Fourth, in a single thread, you should find the top results for all your futures. Since four futures should have generated only a few results, this should be very quick.

Finally, you should print the final five results, along with the state, location name, latitude and longitude for each entry. You may search the location file using a single thread.

To supply the data to the futures, you may use either the queue you developed in programming assignment 1, or ConcurrentLinkedQueue, from Java's libraries. This is the only concurrent data structure you should use from the Java libraries.

Your program must use futures, callables and thread pools to process each thread. It should have some method of providing an input file for each future. File processing must execute in parallel.

Your program must produce correct results, and perform a correct calculation. For example, if you are requested to print the top 5 warmest temperatures in 2015, you may not assume that those results all come from different locations. This complicates processing somewhat.

Please do not use GUI elements for any portion of your program.

Your program should produce output similar to the following. This output is the maximum temperature over all years’ of data.

id=USC00042319 year=2005 month=7 day=20 element=TMAX value=53.9C qflag=

id=USC00042319 latitude=36.4622 longitude=-116.8669 elevation=-59.1 state=CA name=DEATH VALLEY

id=USC00042319 year=2013 month=7 day=1 element=TMAX value=53.9C qflag=

id=USC00042319 latitude=36.4622 longitude=-116.8669 elevation=-59.1 state=CA name=DEATH VALLEY

id=USC00042319 year=2007 month=7 day=7 element=TMAX value=53.9C qflag=

id=USC00042319 latitude=36.4622 longitude=-116.8669 elevation=-59.1 state=CA name=DEATH VALLEY

id=USC00042319 year=1998 month=7 day=18 element=TMAX value=53.9C qflag=

id=USC00042319 latitude=36.4622 longitude=-116.8669 elevation=-59.1 state=CA name=DEATH VALLEY

id=USC00042319 year=1998 month=7 day=19 element=TMAX value=53.3C qflag=

id=USC00042319 latitude=36.4622 longitude=-116.8669 elevation=-59.1 state=CA name=DEATH VALLEY

To reduce the amount of code you must write, you may modify the following code to parse the lines from the station and weather files. Note that the following format is not particularly memory-efficient; it duplicates a lot of data. It might be necessary to make it more efficient.

**class** WeatherData {

String id;

**int** year;

**int** month;

**int** day;

String element;

**int** value;

String qflag;

}

String id = thisLine.substring(0,11);

**int** year = Integer.*valueOf*(thisLine.substring(11,15).trim());

**int** month = Integer.*valueOf*(thisLine.substring(15,17).trim());

String element = thisLine.substring(17,21);

**int** days = (thisLine.length() - 21) / 8; // Calculate the number of days in the line

**for** (**int** i = 0; i < days; i++) { // Process each day in the line.

WeatherData wd = **new** WeatherData();

wd.day = i + 1;

**int** value = Integer.*valueOf*(thisLine.substring(21+8\*i,26+8\*i).trim());

String qflag = thisLine.substring(27+8\*i,28+8\*i);

wd.id = id;

wd.year = year;

wd.month = month;

wd.element = element;

wd.value = value;

wd.qflag = qflag;

}

**class** StationData {

String id;

**float** latitude;

**float** longitude;

**float** elevation;

String state;

String name;

}

StationData sd = **new** StationData();

sd.id = thisLine.substring(0,11);

sd.latitude = Float.*valueOf*(thisLine.substring(12,20).trim());

sd.longitude = Float.*valueOf*(thisLine.substring(21,30).trim());

sd.elevation = Float.*valueOf*(thisLine.substring(31,37).trim());

sd.state = thisLine.substring(38,40);

sd.name = thisLine.substring(41,71);

**Objectives:**

* Follow the assignment directions.
* Create a program that prints the minimum or maximum elements corresponding to the criteria entered.
* Use thread pools, futures, and callables, correctly, and as specified in the assignment.
* Use no locks.
* Read the files concurrently.
* Sift the top answers out of the input concurrently.
* Print the output station and station information correctly.

**Program Submission**

Zip the source files and submit to the dropbox. Gross violations such as missing files or not submitting a zip file will incur penalties. If the program has syntax errors, the grade will be 0: no exceptions.