1 Front Matter

1.1 Collaboration Statement

The last page of the course syllabus contains a statement of the department collaboration policy with regard to projects. Violation of the collaboration policy can result in failure of the course.

1.2 Reminder: Project Hand in Rules

Read and follow the instructions!

1. All project due dates are specified on Brightspace.
2. All projects handed in must be NetBeans projects.
3. Each student starts the semester with 7 free late days. When handing in a project late you must add a comment telling me that you are using some number of late days for the project.
4. If you hand in a project late, you are not allowed to correct and resubmit. This is the only penalty for late submissions.
5. For projects, you will hand in what is specified by individual assignments. Most of the time it will be an entire zipped project folder. Name the archive with ONLY YOUR LAST NAME. Example: Smith.zip ONLY zip archives will be accepted.

1.3 Objectives

As your first project in this course many of the objectives are those things which you are expected to do for every project. Be sure to read and understand the grading criteria for each project.

1. Learn to read and follow directions!
2. The main() method

- Correct usage of command line arguments in main, and setting them in the project properties.
- Correct creation of an object to call the run() method

3. The run() method. Most other methods must be called from the run() method.
4. Correct reading of the file with all exceptions caught and handled.
5. Using formatted strings.
6. Using parallel arrays.
7. Calculating simple sums.

1.4 Grading Criteria:

Grading for this project will be done as follows:

Grade Earned	Requirements
D	main method(s) are the only static methods in any class. main method in the driver checks command line arguments Usage message is correct for the number of required arguments . main method in the driver creates an object of the class and calls a run method. All code handed in is syntactically correct - the program compiles and runs.
C	Everything above plus Declare Arrays to hold the data Read and store the data Complete the activities in Long Term Averages and Records All methods have complete JavaDoc
B	Everything above plus Find the starting indexes for the years and display those Find the first fall frost dates and display those All methods have complete JavaDoc
A	Everything above plus Average First Fall Frost dates by decade and display them. All code is formatted correctly. Blocks are indented correctly. Revision History is complete All methods have complete JavaDoc

NOTE: The requirements for a D project are exactly the same on all projects.
Zip the entire project and upload it on Brightspace.

2 Introduction

In this project you will be using parallel arrays. Those are 1-D arrays where each element [i] belongs to a single record. Read on. Many real programs work with data. Sometimes

lots of data! This data is often stored in text files and must be read, parsed and converted into values usable by your program. In this assignment you are going to write a program to process weather data from the National Climate Data Center. The NCDC maintains a data collection of daily weather observations for more than 19,000 locations in the United States. Some of these have been recorded since the late 19th century. The data used in this assignment was obtained from their web site www.ncdc.noaa.gov/cdo-web.

Hint: Almost everything you need for this project is in the Syntax Review videos in some form.

Create a Project called ClimateChange and download the data file into your project directory. This contains a condensed version of the daily weather data recorded at the Portland International Jetport since 1941. You are going to be analyzing this data to look for evidence of climate change. The start of this file looks like this:

29220 data records

DATE	TMAX	TMIN
01/01/1941	38	25
01/02/1941	32	20
01/03/1941	31	22
01/04/1941	34	25
01/05/1941	32	20
01/06/1941	29	5
01/07/1941	29	-10

Data files come in many different formats and sometimes need different techniques to parse them. This file is very simple. It is organized by line and in fixed width columns. The first three lines are a header describing the size of the file and the data columns. Each subsequent line represents a single day's weather record. The columns are:
DATE - The date of the observation in format MM/DD/YYYY
TMAX - The maximum temperature in Fahrenheit
TMIN - The minimum temperature in Fahrenheit

3 Your Tasks

1. Allocate Arrays to Hold the Data
   The first line in the file says how many data records there are. You will be using a Scanner to process the file. Read that number and then create 5 separate integer arrays of that size for month, day, year, tmax, and tmin. These will hold all the data from the file. This is a lot of data, but your computer can easily hold it in memory.

2. Read and Store the Data in your Arrays
   Next call your Scanner's nextLine() method 3 times to discard the remainder of the first line ("data records") and the next two lines ("date tmax tmin" and the horizonal line below them). Next you are going to read and store all the data into your arrays. The Scanner's default behavior is to break up data at whitespace (spaces, tabs, and newlines), but we also want to break apart the date string which uses forward slashes to separate the month, day and year. Use this code to change the delimiter to also

include forward slashes before starting to read the actual data:

```
fileScnr.useDelimiter("[/ \t\n\r]+");
```

Now you can just use .nextInt() to read each of the five fields: month, day, year, tmax, and tmin. The Scanner will then automatically wrap to the next line and you can continue reading the entire file, and storing it in your arrays.

Verify that you have read in the data correctly. Either examine your arrays using the debugger or print them. Do they match the contents of the file?

3. Long Term Averages and Records

   (a) Write a method that finds and returns the index of the maximum value in a 1-D array that is passed to it.

   (b) Write a method that finds and returns the index of the minimum value in 1-D array that is passed to it.

   (c) Write a method that finds and returns the average (a double) of the numbers in a 1-D array that is passed to it.

   (d) Write a method that finds and returns the first index of a given value in a 1-D array between two indexes that are passed to it.

   Use those methods to find and print:

   - The highest temperature in tmax and the date it occurred on and then use that index for your month[], day[], and year[] arrays)
   - The lowest temperature in tmin and the date it occurred on
   - The average tmax (Check your value. It should be 55.6608...)
   - The average tmin

4. Finding the starting index for each year.
   One aspect of climate change that has been noted in Maine is that the first fall frost has been occurring later in the year. We are going to analyze the minimum daily temperature data to calculate the date each fall when the temperature first reaches freezing temperatures. First we are going to find the starting index of each year in the 'year' array. Your task is to create a loop that prints:

year	starting index
1941	0
1942	365
1943	730
1944	1095
1945	1461

Remember it is poor programming to hard code specific values such as 1941 and 2020 in a program because then the program would have to be modified to process a different file. Instead use year[0] to get the starting year and year[n-1] to get the last year. (I used the variable n for the number of data records.) I suggest that you store these values in a 1-D array. You should be able to figure out how long to make the array. You know how to access the first year and the last value and there are 12 months in each year. Although there might be missing data, you can still make a good guess at the length of the needed array since it is the number of years.

5. Finding the first fall frost
   Make a method
   `int firstFallFrost(int[] tmin, int yearStartIndex, int yearEndIndex)`
   that looks at a one year section of the array for the date of the first fall frost and returns the index of that day. Hint: start searching forward from the middle of the year. For testing, you should find that for 1941 (tmin, 0, 365) it occurs at index 262, and for 1942 (tmin, 365, 730) it occurs at index 636. In the next part you will use the month and day arrays to print the human understandable dates 9/20 and 9/29.

6. Printing the first fall frost dates
   Use your starting year indices from Part 4, your firstFallFrost() method from part 5, and the month and day arrays, to print a table of the first fall frost dates for each year like this:

year	first fall frost
1941	9/20
1942	9/29
1943	10/10
1944	9/25
1945	9/30

   Make sure your results match.

7. Averaging by decade:
   A nicer way to present the data is averaged by decade, i.e. the average fall frost day in the 1940's, 1950's, ... 1920's. Write code to do the analyses this way. Note that for some decades we may not have data for all 10 years (in our data the 1940's and the 2020's). Your code should still give correct averages for these decades. Print your results in a neat table. Use `printf` and format specifiers NOT tabs or spaces to line up your columns.

8. Analysis of the results:
   The first fall frost dates vary a lot from year to year. Compare the average of the first 10 years of the historical record to the average of the most recent 10 years. How much has it changed? Print a simple statement about the change in the average first fall frost date over these 80 years. The following method should be added to your program to do this analysis:

```
/**
 * The day number is the number of days that have elapsed since noon on<br>
 * January 1, 4712 BCE - the beginning of the calendar developed during<br>
 * the reign of Julius Ceasar. For more information .. try Google. <br>
 * This makes it easier to compute differences between two dates or <br>
 * to calculate an age. <br>
 * A = Y / 100 <br>
 * B = A / 4<br>
 * C = 2 - A + B<br>
 * E = 365.25 * (Y + 4716)<br>
 * F = 30.6001 * (M + 1)<br>
 * JD= C + D + E + F - 1524.5 <br>
 */

 private double calcDayNumber(int day, int month, int year){
     double a, b, c, e, f;
     a = year / 100;
     b = a / 4;
     c = 2 - a + b;
     e = 365.25 * (year + 4716);
     f = 30.6001 * (month + 1);
     return  c + day + e + f - 1524.5;
 }
```

With this method in place you can obtain the difference between January first and the frost date using this:

```
int dayNum =  calcDayNumber(day[i], month[i], year[i])
            - calcDayNumber(1, 1, year[i]);
```

4 Partial Example Output

```
run:
The highest temperature in the data is: 103 degrees F which occurred on 08/02/1975
The lowest temperature in the data is: -39 degrees F which occurred on 02/16/1943
The average high temperature was 55.66 degrees F.
The average low temperature was 36.27 degrees F.
Year   Starting Index
Number of years: 80
1941               0
1942             365
1943             730
1944            1095
...
2018           28124
2019           28489
2020           28854

Year        First Fall Frost
1941            09/20
1942            09/29
1943            10/10
...
2020            10/09

Decade         Average DoY    Approx
              of First Frost    Date
1940's            271          08/26
1950's            270          08/25
1960's            272          08/26
1970's            279          09/03
1980's            283          09/07
1990's            280          09/05
2000's            285          09/09
2010's            292          09/16
BUILD SUCCESSFUL (total time: 1 second)
```