

CPS 188 Term Project**Team Leader****Name: Yin Lee Assang****Section Number: 06****Email: yin.leeassang @ torontomu.ca****Team Members****Name: Michael Jantzi****Section Number: 07****Email: michael.jantzi@ torontomu.ca****Name: Kirusanth Palakanthan****Section Number: 08****Email: kpalakanthan @ torontomu.ca****Name: Aden Tsegay****Section Number: 06****Email: aden.tsegay@ torontomu.ca****We affirm that this project is original and is our own work.****Signatures:**

X Kirusanth P.

X Michael J.

X 

X

Table of Contents

Introduction.....	2
Purpose.....	2
Calculations.....	3
Problem 1.....	3
Problem 2.....	4
Problem 3.....	5
Problem 4.....	6
Problem 5.....	7
Graphs.....	8
Problem 6.....	9
Problem 7.....	10
Problem 8.....	11
Problem 9.....	12
Problem 10.....	13
Problem 11.....	14
Conclusion.....	15
Appendix.....	16
Main C Code.....	16
GNUplot Code.....	16

Introduction

The evolving story of Earth's climate is intricately captured in temperature records. This study delves into analyzing a dataset containing both land and ocean temperatures meticulously documented on a basis, from 1750 to 2015. Using the C programming language well known for its efficiency in handling datasets this project aims to unravel the complexities of temperature trends spanning over two and a half centuries. The dataset being explored provides a window into climatic conditions offering an opportunity to examine temperature fluctuations, patterns, and anomalies over time. By employing programming methods and statistical analysis this endeavor aims to reveal patterns that shed light on the context of present climate trends and aid in forecasting future climatic changes.

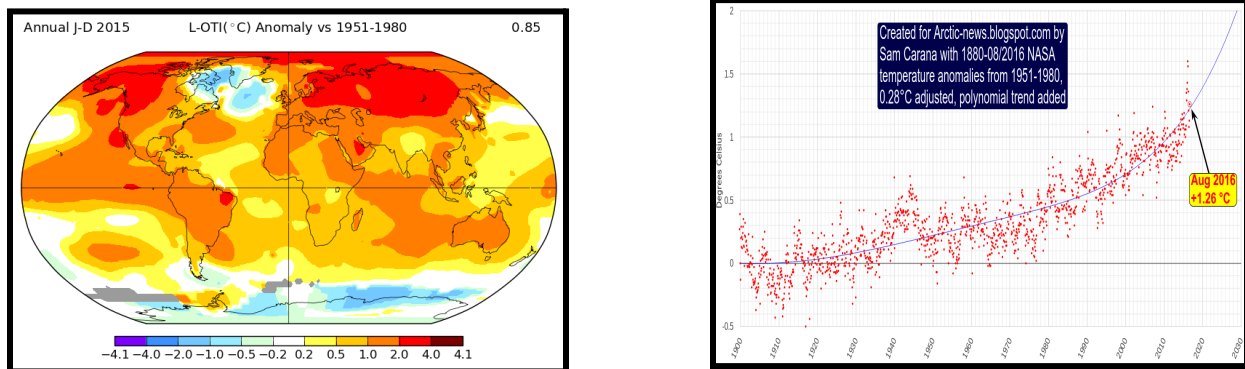


Figure 1.0: Visual representation of how negatively impacted the global temperature is increasing exponentially across centuries.

The main goal of this study is multifaceted; to leverage the power of C programming for the analysis of historical temperature data; to calculate average temperatures detect significant trends and anomalies and to compare temperature variations between land and ocean measurements. Moreover, it contains the land average temperature, land maximum temperature, land minimum temperature, and the land and ocean temperature. This analysis does not showcase programming expertise. Also makes a valuable contribution to understanding climate change—a phenomenon crucial for our planet's future. This project is tailored for an audience comprising the Academic and Research Community, as Educational Institutions. Scholars and researchers, in science, climatology, and related fields will appreciate the data analysis and insights for their further study. Teachers and students at levels can benefit from applying programming skills to real-world environmental data, in educational settings.

Calculations

Problem 1

Based on the land average temperature column, calculate the yearly averages for each year between 1760 and 2015 (the average of the twelve months of each year). One average per year. Ignore the years 1750-1759.

Code Execution: (screenshot of the code output)

Question 1

Year	Avg Temp(C*)
----	-----

1760	7.185167
1761	8.772500
1762	8.606500
1763	7.496750
1764	8.400333
1765	8.251917
1766	8.405667
1767	8.221500
1768	6.781333
1769	7.694583
1770	7.691917
1771	7.853167
1772	8.193500
1773	8.221500
1774	8.772167
1775	9.183083
1776	8.304000
1777	8.256250
1778	8.542250
1779	8.983250
1780	9.432917
1781	8.102583
1782	7.901250
1783	7.680833
1784	7.862000
1785	7.363000
1786	8.258167
1787	8.031833
1788	8.447167
1789	8.334167
1790	7.982333
1791	8.234250
1792	8.089000
1793	8.229167
1794	8.526000
1795	8.350333
1796	8.271000
1797	8.510583
1798	8.670250
1799	8.505750
1800	8.484250
1801	8.589667

1970	8.700917
1971	8.599250
1972	8.499583
1973	8.948250
1974	8.467500
1975	8.744833
1976	8.347250
1977	8.850250
1978	8.692750
1979	8.733417
1980	8.980333
1981	9.165833
1982	8.639167
1983	9.028167
1984	8.691833
1985	8.658000
1986	8.833583
1987	8.994417
1988	9.201583
1989	8.922000
1990	9.234167
1991	9.179417
1992	8.836583
1993	8.866583
1994	9.038750
1995	9.347083
1996	9.038917
1997	9.202583
1998	9.522667
1999	9.285083
2000	9.201167
2001	9.414583
2002	9.570417
2003	9.525583
2004	9.324583
2005	9.700917
2006	9.532500
2007	9.732167
2008	9.431750
2009	9.505250
2010	9.703083
2011	9.516000
2012	9.507333
2013	9.606500
2014	9.570667
2015	9.831000

Output:

Average temperature for 1760 is 7.185
Average temperature for 1761 is 8.772
Average temperature for 1762 is 8.607
Average temperature for 1763 is 7.497
Average temperature for 1764 is 8.400
Average temperature for 1765 is 8.252
Average temperature for 1766 is 8.406
Average temperature for 1767 is 8.222
Average temperature for 1768 is 6.781
Average temperature for 1769 is 7.695
Average temperature for 2006 is 9.532
Average temperature for 2007 is 9.732
Average temperature for 2008 is 9.432
Average temperature for 2009 is 9.505
Average temperature for 2010 is 9.703
Average temperature for 2011 is 9.516
Average temperature for 2012 is 9.507
Average temperature for 2013 is 9.607
Average temperature for 2014 is 9.571
Average temperature for 2015 is 9.831

Purpose: To demonstrate statistics that provide a deeper understanding of how the average temperature, in Celsius, has varied across each year between 1760 and 2015. This encompasses the impact on a level, during historical periods.	Conflicts: When it comes to difficulty, per question question 1 was the one that received the attention. The main challenge with this question revolved around reading data. This was a bit tricky because there were methods to read files and it took some time to figure out the best approach. Another issue that arose was with arrays. Transferring the data, from the file into	Outputs/Analysis: The output reveals the details, about how the average temperature in Celsius fluctuated between 1750 and 2015 starting at 7.18 and reaching 9.83. Upon analyzing the provided information it becomes evident that the yearly averages from 1760 to 2015 exhibit a pattern of alternating increases and decreases than remaining constant. However, there is a	How we would approach next time: A straightforward method of coding for this issue next time would be advantageous, along, with developing functions to enhance the program. In particular, utilizing array pointers to allocate the percentage values would be helpful.
--	--	---	--

	those arrays also posed some difficulties	rise of 2.64 degrees, between the final years.	
--	---	--	--

C operations we used to answer question #1 (Explanation of code)

Variable Declarations, at the Beginning:

- Initializing variables such as count i and n. The count keeps track of the processed months for the current year while I acts as an index for storing yearly averages.
- Defining a character array temp[MAX_LENGTH] to hold each line read from the file.
- Using pointer variables token, monthlyavg and year for string manipulation purposes after splitting the line.
- Utilizing floating point variables like currentyearlyavg and yearaverage[255] to calculate the average store an array of yearly averages, dmonthlyavg for monthly average as a double centuryavg[MAX_LENGTH] century, along with other unused variables within this excerpt.

Reading and Initializing the file:

1. File Access and Line Reading;

Assuming that the file has already been opened (omitted we employ fgets to read each line into the temp buffer.

2. Parsing Individual Lines;

Anticipating data in a format, like "year month temperature " we employ strtok to separate out the year and monthly average temperature. The year and month are separated by a dash. Only the year is relevant here.

3. Filtering and Data Conversion;

The year extracted from the line is changed from a string to an integer using atoi. The process continues only if the year falls within the specified range (from 1760).

The average monthly temperature string is converted to a double precision floating point number using atof.

4. Calculating Averages;

For each entry the monthly average temperature is added to currentyearlyavg and count is increased. When count reaches 12 (indicating processing of data for a year) the code calculates the average temperature by dividing currentyearlyavg by 12.

The annual average is then saved in the yearaverage array. Counters are reset, for the year.

Finalizing and Outputting Results;

The loop runs until reaching the end of file (`feof(file)` returns true).

Once all data has been processed the file is closed with `fclose(file)`.

Finally a header is printed out followed by displaying each year (starting from 1760) alongside its corresponding calculated temperature stored in the `yearaverage` array

Problem 2

Based on the land average temperature column, calculate the average land temperature for the different centuries: 18th century (1760-1799), 19th century (1800-1899), 20th century (1900-1999) and 21st century (2000-2015). One average per century.

Code Execution: (screenshot of the code output)

```
Question 2

Century    AverageTemperature
1760-1799   8.214998
1800-1899   8.009105
1900-1999   8.637712
2000-2015   9.542094
```

Output:

18th century temperature average is
8.215

19th century temperature average is
8.009

20th century temperature average is
8.638

21st century temperature average is
9.542

Purpose:	Conflicts:	Outputs/Analysis:	How we would approach next time:
To demonstrate statistics that provide a deeper understanding of how the average temperature, in Celsius, has varied across various centuries between the 18th to 21st. This encompasses the impact on a level, during historical periods.	When it came to the challenge, merging questions 1 and 2 in a single file turned out to be more difficult than expected. Some other conflicts encountered during this process include setting up variables like "centuryavg[]". Yearaverage[]". Additionally the code tries to divide "century" by "count" without ensuring that "count" is not zero. This leads to a division by zero error if the first year checked ("i % 100 == 0")	The output reveals the details, about how the average temperature in Celsius fluctuated between the century 18th to 21st centuries. Upon analyzing the provided information it becomes evident that the average land temperature for the different centuries exhibits a pattern of alternating increases and decreases rather than remaining constant. However, there is a rise of 2.64 degrees, between the 18th and 21st century is approximately 1.32.	A comparable method could be used next time through the use of effective techniques for determining the average land temperature over various centuries would be utilized preferably.

	<p>does not result in an increase of "count". Moreover managing types and declarations posed a challenge making sure all variables, like 'yearaverage' 'i' 'count' are properly declared with types such as int, float and double. These are some mistakes encountered along with others like century average calculation, for the first century.</p>		
--	---	--	--

C operations we used to answer question #2 (Explanation of code)

Setting Up Initial Values:

We set count and n to 0. Count is used to track the number of years considered in calculating the temperature for each century while n serves as an index, for storing these calculated temperatures into the centuryavg array.

The line `printf("\nQuestion 2\n\n");` simply displays a title. Heading for this section of output.

Displaying Column Headings;

Next it prints "Century AverageTemperature\n" to show column titles that will be used in the output indicating each century alongside its temperature.

Looping for Temperature Calculation:

A for loop runs from 1760, to 2016 ($1760 + 256$) computing the temperature for each century within this time frame. Within this loop if ($i \% 100 == 0 \parallel i == 2016$) checks if a year marks either a centurys end or reaches the data ranges endpoint (2016). It's worth mentioning that when a year is divisible, by 100 (like 1800 or 1900) it signals the conclusion of a century. Once we reach the end of a century (. In the case of 2016, which marks the end of our data range) we divide the accumulated temperature for that century by the count (representing the number of years within that century) to determine the average temperature for that period. The resulting average temperature is then saved in the array called centuryavg[n]. A printf statement is used to display both the range of years for that century (1760 1799) and its corresponding average temperature. Following this we reset the variables count and century in preparation for calculating averages for the century. Specifically n is incremented to move to the position in the centuryavg array count is reset to zero to start counting years in the century and century is reset to zero for accumulating temperature values anew. In each iteration, through our loop representing each year we add up each years temperature (stored as yearaverage[i 1760]) to our tally for that century under consideration. This process helps us sum up all temperatures across each year within a given century.

Additionally incrementing `count++` ensures we keep track of how years contribute towards calculating an average for our ongoing century.


Initialization steps are taken into account for Monthly Averages:

After computing century averages the program sets up a `monthlyaverage` array, with 12 slots to hold the temperatures for each month. However it doesn't actually use this array for any calculations in the code provided. Additionally it creates a `months` array containing the names of all twelve months for linking temperatures to specific months, in later sections of the program.

Problem 3

Based on the land average temperature column, calculate the monthly averages for each month for all years combined between 1900 and 2015. A total of twelve averages. One average per month.

Code Execution: (screenshot of the code output)

A screenshot of a terminal window with a black background and white text. The text shows the output of a code execution for 'Question 3', displaying a table of monthly average temperatures from 1900 to 2015.

Month	Avg Temp
January	2.815
February	3.331
March	5.396
April	8.537
May	11.395
June	13.540
July	14.449
August	13.958
September	12.167
October	9.527
November	6.223
December	3.812

Output:

Monthly Averages between 1900 -

2015: January: 2.30

February: 3.00

March: 4.97

April: 8.24

May: 11.13

June: 13.31

July: 14.29

August: 13.74

September: 11.71

October: 8.94

November: 5.75

December: 3.22

Purpose: To showcase the breakdown of	Conflicts: In question 3 the main conflict we	Outputs/Analysis: The output reveals the details, about how	How we would approach next time: One way to improve
---	---	---	---

calculations, for the monthly averages based on each month according to the land average temperature column across the years spanning from 1900 to 2015.	encountered was figuring out a method to calculate the average using the land temperature data from 1900 to 2015. We also struggled with defining variables like 'MAX_LENGTH' and 'temp' as creating arrays of strings without specifying their size. Additionally, we found it tricky to deal with tasks, like file handling dividing by magic numbers such as 116, and using feof().	the monthly averages based on each month from the land average temperature fluctuated between 1900 to 2015. Upon analyzing the provided information it becomes evident that the output exhibits a pattern of alternating increases and decreases rather than remaining constant. However, there is a rise of 0.997 degrees, between 1900-2015, monthly land average temperature.	the efficiency of the code is to reduce the reliance, on if statements. Utilizing loops (, for/while) and switch statements could prove advantageous.
--	--	--	---

C operations we used to answer question #3 (Explanation of code)

For this problem, the monthly averages were to be determined using the land average temperature columns for the years 1900 to 2015. The code starts off with opening and reading the file "GlobalTemperatures.csv" as previously stated, extracting the date (year-month) and the two land average temperatures from each line and storing it in an array. Processing from the years 1900 to 2015, the code then initializes the variables for each month of their respective years and simply calculates the average of the two temperatures from the data. In total there should be a total of twelve average monthly temperatures outputted for each year written in a text file, "AverageMonthlyTemperatures.txt".

Variable Declarations

An array named "monthlyaverage" is created to hold the temperatures for each month of the year. Since there are 12 months in a year the array size is set to 12. The array "months" is defined as a two character array where each row stores the name of a month. The length of each month name is limited by the MAX_LENGTH defined elsewhere, in the code.

Reading and Processing Data from File

The file named "GlobalTemperatures.csv" is opened in read mode ("r"). A loop using do while is implemented to read each line from the file until reaching the end (feof(file)). For every line; fgets(temp, MAX_LENGTH, file); reads a line from the file into a buffer called temp. strtok(temp, " "); is applied to elements in temp based on commas. The first strtok call retrieves the token (likely representing a date, in YYYY MM DD format). Isn't directly utilized. monthlyavg = strtok(NULL, " "); fetches the token likely

representing the average temperature as a string. The program first breaks down the date token into parts using a hyphen to extract the year component ensuring that it is equal, to or greater than 1900.

When there is a temperature value, indicated by `monthlyavg` not being `NULL` it gets converted to a double. Then added to the sum of temperatures for that particular month in `monthlyaverage[count]`. The count variable increments through the months resetting back to 0 once it reaches 12. Certain variables such, as `count`, `temp`, `token`, `monthlyavg`, `year`, `file` and `i` need to be declared in the code. Additionally it assumes the presence of a called `MAX_LENGTH` which should be defined elsewhere in the code for it to compile and execute correctly.

Problem 4

Based on the land average temperature column, what was the hottest month recorded and what was the coldest month recorded (month and year in each case). In case of a tie, mention only one (doesn't matter which one).

Code Execution: (screenshot of the code output)

```
Question 4
The hottest month was on July 1761 at 19.02
The coldest month was on January 1768 at -2.08
```

Output:

Hottest month recorded:

Month: July, Year: 1761, Temperature: 19.02 degrees
celsius Coldest month recorded:

Month: January, Year: 1768, Temperature: -2.08 degrees celsius

Purpose: The main goal of this program is to identify the coolest and hottest months documented using the land's average temperature data for a specific location.	Conflicts: Determining the coldest months along, with their years based on the average temperature of a specific region proved to be quite challenging. This situation resembled the dilemma. With the added complication of factoring, in an extra variable.	Outputs/Analysis: The results show information, about the hottest and coldest months of the year determined by the land's average temperature, on land. Upon reviewing the data it is clear that the report confirms that July 1761 was the hottest month and January 1768 was the coldest.	How we would approach next time: The method would be quite similar since it is a question, with only a few approaches to think about; simple and direct.
--	---	---	--

C operations we used to answer question #4 (Explanation of code)

The second question followed a method, to the question operations but it necessitated extra arrays for identifying the hottest and coldest months. These arrays were used to calculate the land temperature utilizing the index numbers of the lowest values to pinpoint the warmest and coolest months as well, as the corresponding year.

Problem 5

Based on your answer in question 1, what year was the hottest and what year was the coldest?

Code Execution: (screenshot of the code output)

```
The coldest Land Average Temperature is 6.781333 in 1768
The hottest Land Average Temperature is 9.831000 in 2015
```

Output:

The hottest year is 2015

The coldest year is 1768

Purpose: The main goal of this program is to identify the coolest and hottest years documented using the land's average temperature data for a specific location.	Conflicts: The main challenge encountered in the question was figuring out a method to identify the coldest years based on the average temperature of the land. It wasn't sufficient to compare each land's temperature, with the national average separately. A separate array, mirroring that of question 1 needed to be generated to facilitate comparisons based on array positions.	Outputs/Analysis: The results show information, about the hottest and coldest year determined by the land's average temperature, on land. Upon reviewing the data it is clear that the report confirms that the coldest land average temperature is 6.781333 in 1768 and the hottest land average temperature is 9.831000 in 2015.	How we would approach next time: The method would be quite similar since it is a question, with only a few approaches to think about; simple and direct.
---	--	--	--

C operations we used to answer question #5 (Explanation of code)

The analysis section kicks off with a header labeled as Question 5 to mark the beginning of this part. It sets up two groups of variables, hyear and cyear initially at 0 to hold the years with the lowest temperatures. The initial values, for htemperature and ctemperature are both set to the

value in `yearlyAverages[0]` assuming that `yearlyAverages` contains the temperatures from 1760 to 2016 with the first entry representing 1760.

Moving through the years;

Using a for loop that runs 256 times (i from 0 to 255) covering the years from 1760 to 2016. The loop goes up to 255 because arrays start at index zero in C and there are a total of 257 years counted from zero. During each iteration it compares if the average temperature of the year (`yearlyAverages[i]`) is greater than `htemperature`. If it is `htemperature` gets updated with this value and `hyear` is assigned as `i`. 1760 representing that year.

Likewise it also checks if the average temperature of the year is less, than `ctemperature`. If that's the case the temperature gets updated to the value. The year is then set to match.

Displaying the outcomes;

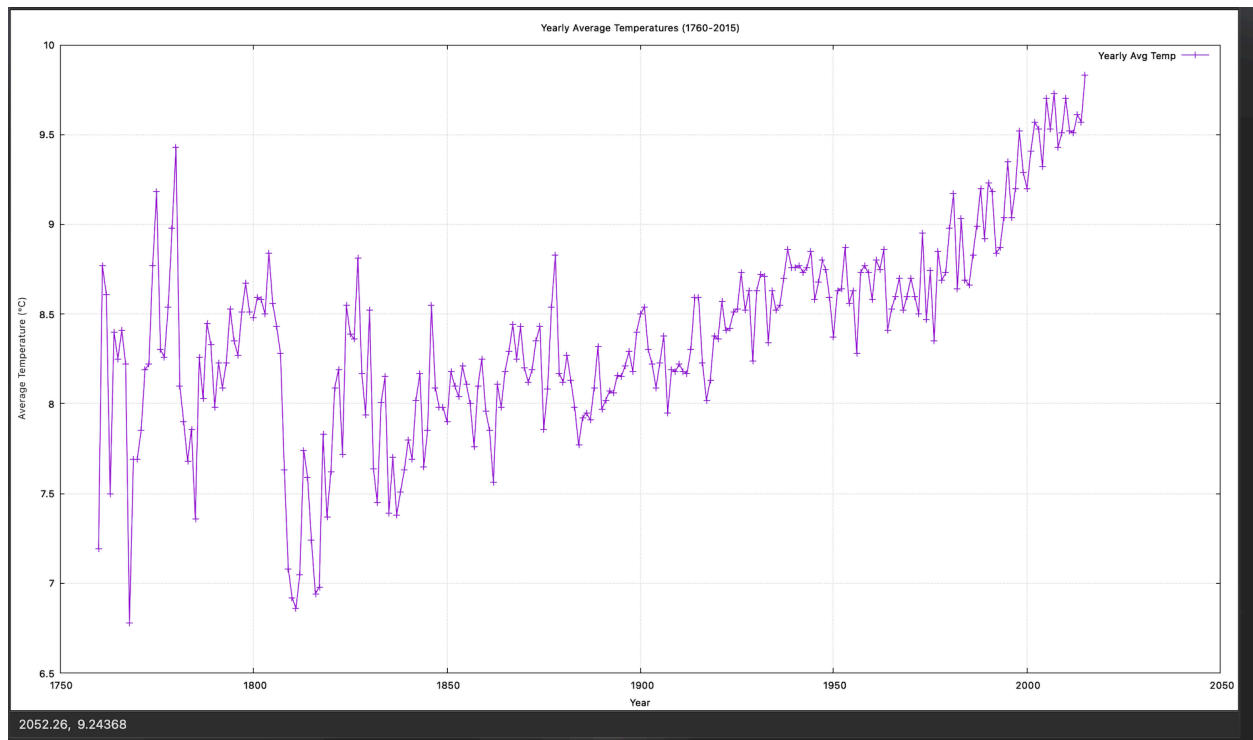
Once the loop ends the code showcases the lowest temperatures recorded in years along, with their average values using `printf`. The placeholder `%d` is employed for integer variables (year and current year) while `%lf` represents variables (highest temperature and current temperature) assuming that yearly averages are stored as doubles. If yearly averages are stored differently (e.g., as floats) `%f` should be used of `%lf`.

Graphs

Problem 6

Based on your answer in question 1, generate a GNUPlot data file and use GNUPlot to make a graph (line plot) of the yearly temperatures for the years 1760 to 2015. Label the axes clearly and add a title and legend to your graph.

Plot (screenshot of the plot)



GNU PLOT SCRIPT FOR QUESTION 6:

```
gnuplot> set title "Yearly Average Temperatures (1760-2015)"
gnuplot> set xlabel "Year"
gnuplot> set ylabel "Average Temperature (°C)"
gnuplot> set grid
gnuplot> plot "avgTemp.txt" using 1:2 with linespoints title "Yearly Avg Temp"
```

Purpose:

This linear graph plot

Conflicts:

There were minor

Outputs/Analysis:

The result displays a

How we would

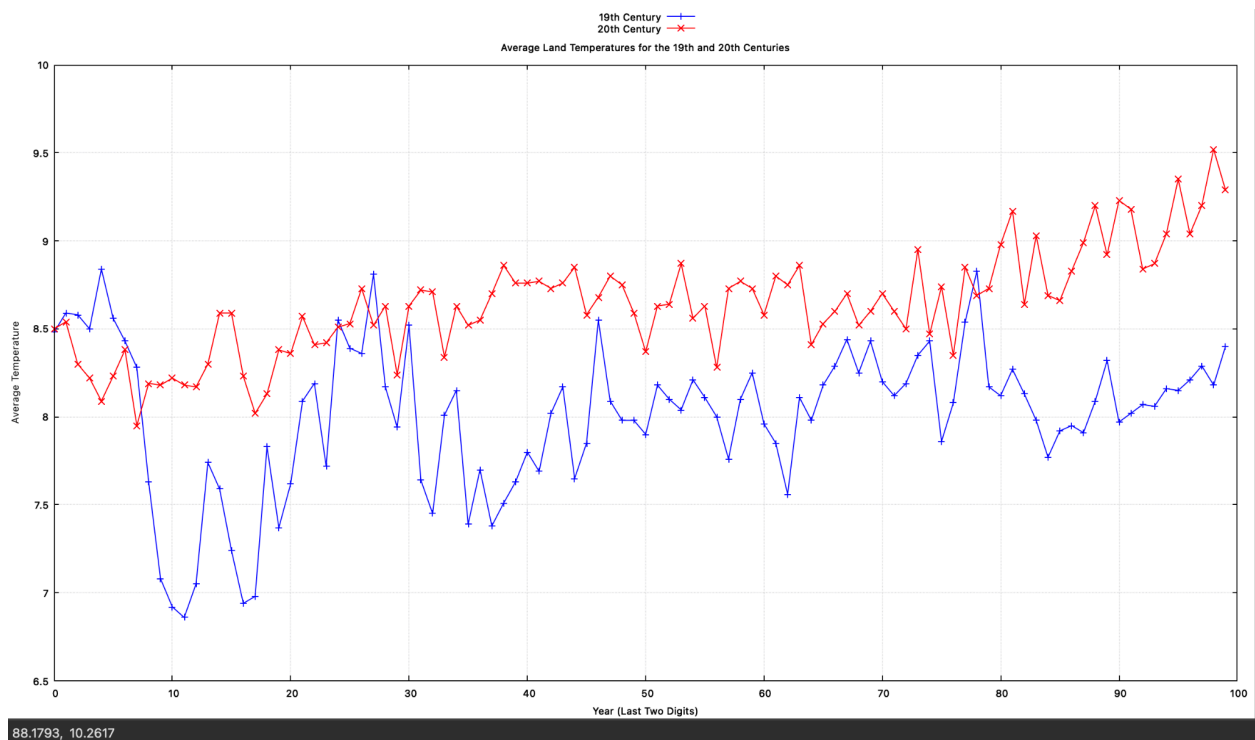
approach next time:

<p>illustrates a visual representation of the yearly temperature for the following years of 1760-2015. Its primary aim is to provide a depiction of the temperature variations, over a span of 255 years spanning various centuries.</p>	<p>problems, with coding for GNUplot in this particular scenario. Specifically incorporating lines of the yearly temperature for the following years; 1760-2015, on the graph posed a challenge. Additionally setting up the for loop to input the values into the file proved to be somewhat troublesome.</p>	<p>line plot depicting lines representing the temperatures from 1760 to 2015. The pattern, in this graph, alternates in a pattern quickly resembling an exponential curve overall. To improve the presentation the axes were annotated with titles and a legend.</p>	<p>Ultimately, when approaching this question the steps to consider would remain unchanged as it is a query, with options to explore.</p>
--	--	--	---

Problem 7

Generate a GNUPlot data file and use GNUPlot to make a graph (line plots) of the average land temperatures for the 19th and 20th centuries. Put both lines on the same figure. Ensure that you have the same x-axis scale (for example 1852 and 1952 would both have an x-value of 52). Have your two line plots with different colours. Label the axes clearly and add a title and legend to your graph.

Plot (screenshot of the plot)



GNU PLOT SCRIPT FOR QUESTION 7:

```
gnuplot> set datafile separator '\t'  
gnuplot> set title "Average Land Temperatures for the 19th and 20th Centuries"  
gnuplot> set xlabel "Year (Last Two Digits)"  
gnuplot> set ylabel "Average Temperature"  
gnuplot> set key outside top center  
gnuplot> set grid  
gnuplot> plot 'Q7avgTemp.txt' using (int($1)%100):2 with linespoints lt rgb "blue" title '19th  
Century', 'Q7avgTemp.txt' using (int($3)%100):4 with linespoints lt rgb "red" title '20th Century'
```

<p>Purpose: The linear graph plot aims to provide a representation of the average land temperatures, in the 19th and 20th centuries. Upon reviewing the multiple line plots graph it is apparent that there is a positive correlation between the land temperature and the century as both show a gradual increase with an alternating pattern, in temperature fluctuations.</p>	<p>Conflicts: The major challenge with this task was revisiting the process of reading the string data from the GlobalTemperatures.csv file and transforming it into data for plotting in GNUplot. It became particularly challenging given that the code from tasks was left incomplete. Another hurdle was setting up multiple line plots that intersect with each other along, with determining the range to ensure the graph is visually pleasing and easily understandable.</p>	<p>Outputs/Analysis: Upon analyzing the two different line plots, we can observe the positive correlation between the land temperatures of the 19th and 20th centuries. From the representation provided it is evident that during the century there is a more consistent trend in average land temperature with minimal fluctuations. In contrast, the 19th century exhibits a fluctuating pattern resulting in surpassing average land temperatures from the 20th century. In essence, this multiple-line plot graph illustrates the relationship, between the land temperatures of both periods.</p>	<p>How we would approach next time: Ultimately, when approaching this question the steps to consider would remain unchanged as it is a query, with options to explore.</p>
---	---	--	---

Problem 8

Using the columns LandAverageTemperature, LandMaxTemperature and LandMinTemperature, generate a GNUPlot data file and use GNUPlot to make line plots that show all three temperatures on the same figure. Use the years for the x-axis (use only the years between 1850 and 2015) and the yearly averages for the y-axis. Use three different colours (or different line styles) for the three lines. Make sure that the line plotting LandAverageTemperature stands out from the other two (ex: make that line thicker). Make sure your graph has a title, axes labels and a legend that explicitly tells which line is which.

Plot (screenshot of the plot)

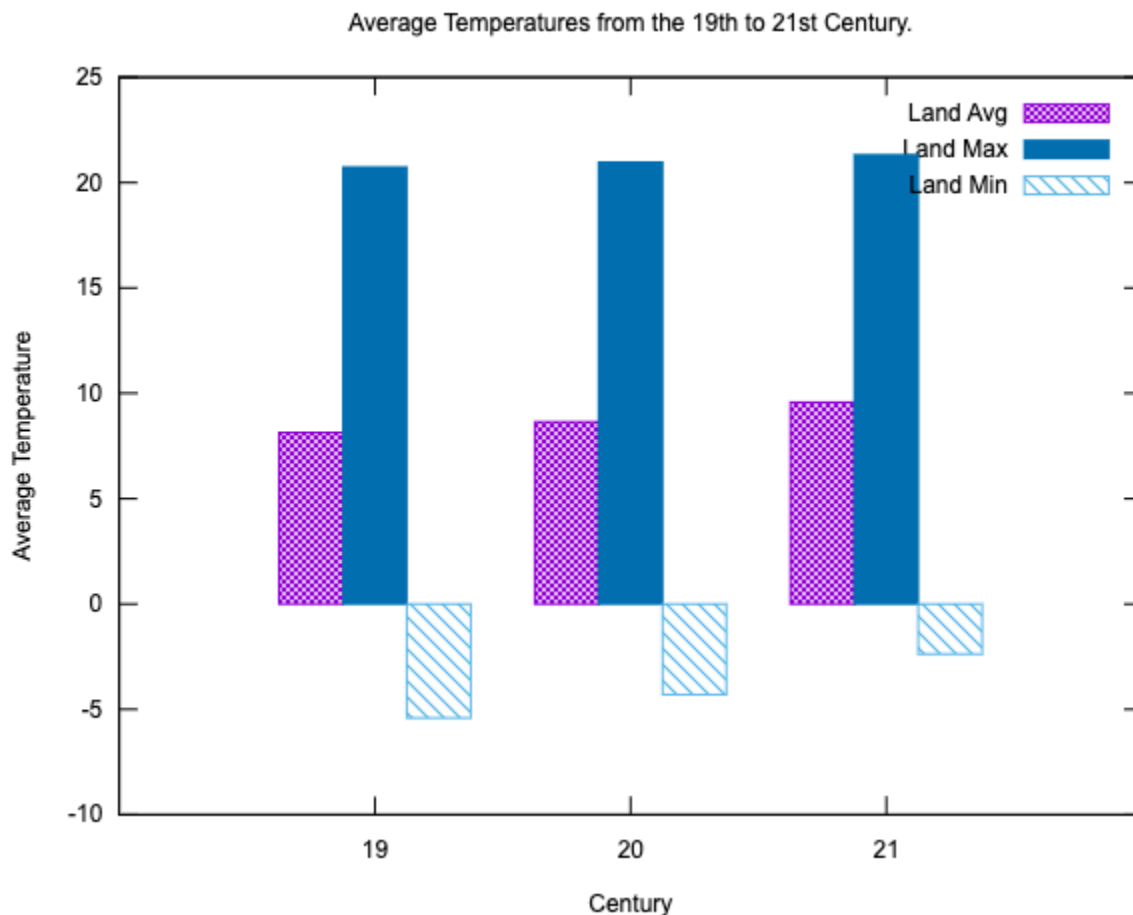
GNU PLOT SCRIPT FOR QUESTION 8:

<p>Purpose: The line plot graphs aim to provide a representation of the average land temperatures, land max temperatures, and land min temperatures between the years 1850 and 2015. The main goal is to generate a GNU plot where years are the x-axis and the yearly axis is the y-axis. Through analyzing the three different line plots, it should be evident the line plotting average temperatures stands out more presentably than others.</p>	<p>Conflicts: The major challenge with this task was revisiting the process of reading the string data from the GlobalTemperatures.csv file and transforming it into data for plotting in GNUplot. It became particularly challenging given that the code from tasks was left incomplete. Another hurdle was setting up multiple line plots that intersect with each other along, with determining the range to ensure the graph is visually pleasing and easily understandable.</p>	<p>Outputs/Analysis: After examining the three different line plots of min average, min average, and average temperature, it is apparent that there are alternating trends where the temperature ranges show an increase, between the years 1850 and 2015. The graphs clearly indicate a relationship between the minimum and maximum temperatures dating back to 1850 while the average temperature fluctuates, across the plot beginning in 1750.</p>	<p>How we would approach next time: Ultimately, when approaching this question the steps to consider would remain unchanged as it is a query, with options to explore.</p>
--	---	--	---

Problem 9

Using the columns LandAverageTemperature, LandMaxTemperature and LandMinTemperature, generate a GNUPlot data file and use GNUPlot to make three bar plots (box or histogram plos) that show the average, low and high temperatures for each of the four centuries. Put all 3 century plots on the same figure displayed as subplots (multiplots). Each plot will show the boxes with different colours (one colour per century). Have a title to your figure and have the name of the century on each subplot. Have a legend on each subplot.

Plot (screenshot of the plot)



GNU PLOT SCRIPT FOR QUESTION 9:

```
set title 'Average Temperatures from the 19th to 21st Century.'
set xlabel 'Century'
set ylabel 'Average Temperature'
set style data histogram
set style fill pattern 2
set style histogram cluster gap 1
```

set key top right

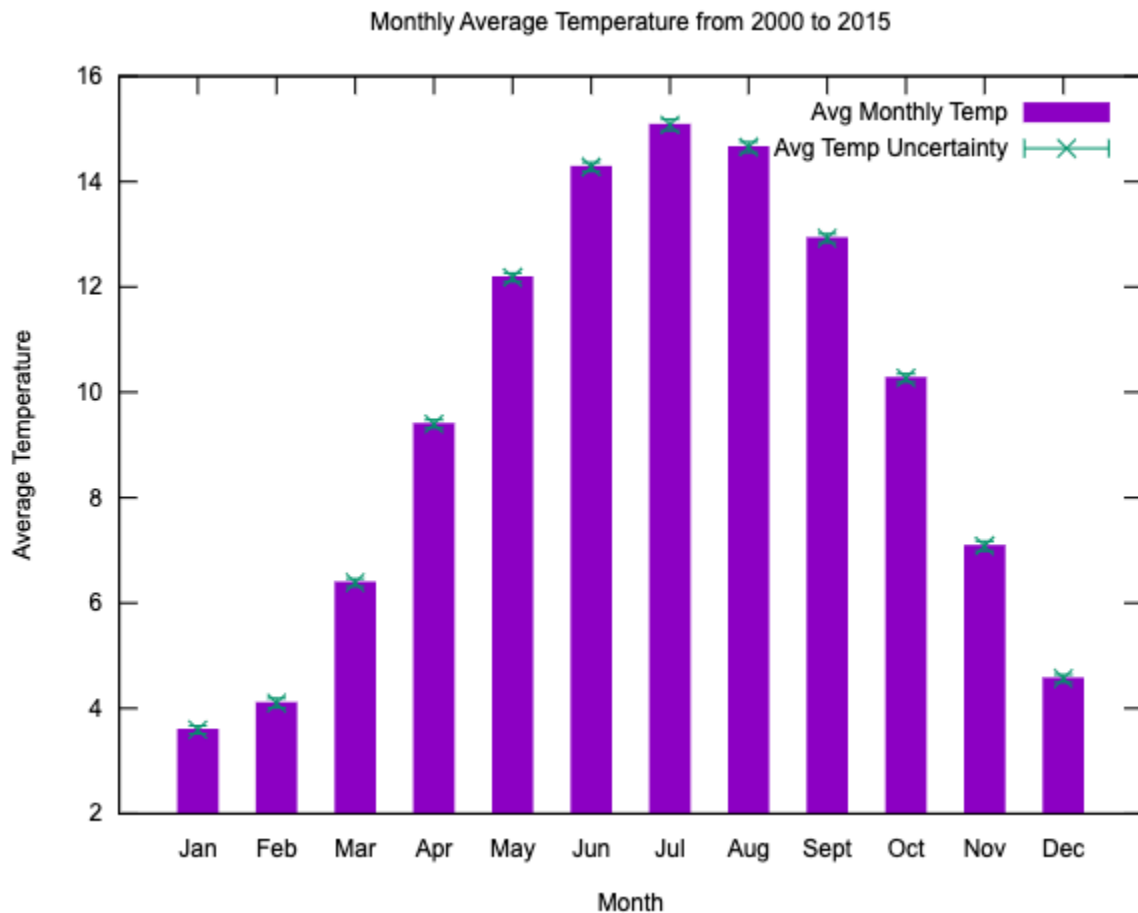
```
plot 'TempsCent.txt' using 2:xtic(1) ls 1 title 'Land Avg', \
'TempsCent.txt' using 3 ls 6 title 'Land Max', \
'TempsCent.txt' using 4 ls 11 title 'Land Min'
```

Purpose: The set of three bar graphs intends to showcase the land temperatures, highest land temperatures and lowest land temperatures illustrating the minimum and maximum temperatures, across each of the four centuries. The primary objective is to create a GNU plot featuring all four century plots, in one figure displayed as subplots. Each plot will exhibit boxes with titles and legends on every subplot.	Conflicts: The major challenge with this task was revisiting the process of reading the string data from the GlobalTemperatures.csv file and transforming it into data for plotting in GNUplot. It became particularly challenging given that the code from tasks was left incomplete. Another hurdle was setting up multiple bar plots for the different average temperatures throughout the century with determining the range to ensure the graph is visually pleasing and easily understandable.	Outputs/Analysis: Upon reviewing the three bar charts displaying average maximum average and overall temperature variations reflecting the average, low and high temperatures it is evident that there are noticeable upward patterns showing a gradual rise in temperature over the course of four centuries. The visual representations clearly suggest a correlation, between the minimum and maximum temperatures indicating trends. This consistent pattern effectively illustrates the temperature variances throughout the centuries.	How we would approach next time: Ultimately, when approaching this question the steps to consider would remain unchanged as it is a query, with options to explore.
--	--	--	---

Problem 10

For the years 2000 to 2015, generate a GNUPlot data file and use GNUPlot to make an error bar plot of the average land temperature by month. Use the uncertainty column for land temperatures to draw the error bars.

Plot (screenshot of the plot)



GNU PLOT SCRIPT FOR QUESTION 10:

```
set title 'Monthly Average Temperature from 2000 to 2015'
set xlabel 'Month'
set ylabel 'Average Temperature'
set xtics 1
set style data histogram
set style fill solid
set style histogram cluster gap 1
```


plot "Q10 data.txt" using 2:xtic(1) title "Avg Monthly Temp",\
 "Q10 data.txt" using 2:3 with errorbars title "Avg Temp Uncertainty"

<p>Purpose: The linear plot graph set aims to display the land temperature for each month from 2000 to 2015. The main goal is to generate a GNU plot illustrating the error bars, for the temperature averages utilizing the uncertainty data in the land temperature columns to represent these errors.</p>	<p>Conflicts: The major challenge with this task was revisiting the process of reading the string data from the GlobalTemperatures.csv file and transforming it into data for plotting in GNUplot. It became particularly challenging given that the code from tasks was left incomplete. Another hurdle was setting up error bar plots for the different average temperatures throughout the years 2000 to 2015 with determining the range to ensure the graph is visually pleasing and easily understandable.</p>	<p>Outputs/Analysis: After examining the error bar graphs from 2000, to 2015 it's clear that there are ups and downs in the land temperature over this 15 year period. The visual data indicates trends, including negative and fluctuating patterns. In terms the shape of this error bar plot resembles a compressed wave like sinusoidal graph with its lowest point around the second year and its highest point around the 15th year, in 2015. This particular GNU error bar plot displays quite a bit of variability.</p>	<p>How we would approach next time: Ultimately, when approaching this question the steps to consider would remain unchanged as it is a query, with options to explore.</p>
--	---	---	--

Problem 11

Generate a GNUPlot data file and use GNUPlot to do a plot similar to what you did in question 6 but only for the years 1850 to 2015 and add the data for the **land** and **ocean** average temperatures columns. Have the two lines on the same figure. Label the axes clearly and add a title and legend to your graph.

Plot (screenshot of the plot)

GNU PLOT SCRIPT FOR QUESTION 11:

Purpose: The goal of the linear plot graph setup is to showcase a GNUPlot question 6. This time covering the years 1850 to 2015 and including data, for both land and ocean average temperatures. In this GNUPlot graph we plan to incorporate two linear plot lines within the visual, ensuring proper labeling of the axes and adding a title and legend to enhance clarity.	Conflicts: There were minor problems with coding for GNUplot in this particular scenario. Specifically incorporating lines of the yearly temperature for the ocean and land columns of the following years; 1850-2015, on the graph posed a challenge. Additionally setting up the for loop to input the values into the file proved to be somewhat troublesome.	Outputs/Analysis: The result displays a line plot depicting lines representing the land and ocean average temperatures from 1850 to 2015. The pattern, in this graph, alternates in a pattern quickly resembling an exponential curve overall. To improve the presentation the axes were annotated with titles and a legend.	How we would approach next time: Ultimately, when approaching this question the steps to consider would remain unchanged as it is a query, with options to explore.
--	--	--	---

Conclusion

In conclusion, after finishing a term project in C coding that involved analyzing and displaying temperature data from 1750 to 2015 some insight was gathered from a programming standpoint. The project aimed to calculate the land temperature on a yearly basis, compute monthly averages for each month, identify the hottest and coldest months and more. In this reflection, what was learnt will be discussed from the analysis of the experience working with C and GNUPlot as considerations for future projects. The visual representations created likely depicted a trend of increasing temperatures over the course of 265 years with an acceleration in warming rates seen in recent decades. This observation aligns with the accepted consensus on global warming, which attributes much of it to human activities. Through data analysis we have not only uncovered long term trends but also fluctuations in temperatures such as seasonal changes and extreme weather occurrences. These patterns play a role in understanding the intricacies of climate systems and forecasting developments. While utilizing C for data processing proved effective it was not without its challenges—particularly concerning memory management and selecting data structures. These difficulties highlighted the need for planning and a deep understanding of C's complexities when dealing with datasets. Debugging code, in C and ensuring data processing and visualization were aspects of the project that emphasized the significance of thorough testing to ensure reliable outcomes. In projects enhancing the use of data structures and algorithms might boost effectiveness and adaptability particularly when dealing with extensive datasets or intricate analyses. To sum up, the project was not about practicing C programming and GNUPlot visualization. Also delved deep into climate data providing valuable perspectives on past temperature trends and patterns. The knowledge skills honed a set of groundwork for future endeavors highlighting the significance of computational analysis, in environmental sciences

Appendix

Main C Code

References

Carana, Sam. “Temperature rise from 1750 to 2016.” *Arctic News*, Arctic News, (n.d.),
<https://arctic-news.blogspot.com/p/temperature.html>