Toronto Metropolitan University Computer Science

# Introduction

The purpose of this report is to analyze the prevalence of diabetes in Canada's four most populated provinces (Ontario, Quebec, British Columbia, and Alberta) between the years 2015 and 2021. The report is based on accurate data collected by Statistics Canada, which provides valuable insights into the health status of the Canadian population.

The report presents a comprehensive analysis of the data, including calculations of various averages, comparisons between provinces and national standards, and identifying trends over time. The report also includes visual representations of the data in the form of graphs and charts generated using GNUPlot functionalities.

The report is presented as a single program written in C language (code located at the end of the report ), with each section addressing specific questions related to the prevalence of diabetes in the four provinces and the country as a whole. The program includes text comments to identify which question is answered in each section, making it easy to navigate and understand. The following section of the report will provide a brief explanation of the methodology used to answer each question, along with a screenshot of the output for that specific question.

Overall, this report aims to provide valuable insights into the prevalence of diabetes in Canada's four most populated provinces, helping to inform public health policies and initiatives to reduce the impact of this chronic disease on the Canadian population.

# Additional Function Explanations

## format(Burgundy)

The “format()” function is used to go through the raw data given to us by the professor and process it into data that is easy to read and use throughout the program. When executing this code, two files are opened: the raw data file “statsscan_diabetes.csv” and a text file to store the formatted data “formattedData.txt”. With a “while” loop that does not end until all the characters in the raw data are gone through, the function scans each character from the raw data until it hits a quotation mark (“) character. This character signifies the beginning of a value in the raw data and triggers an “if” statement that opens another “while” statement that stores all the characters of the value (skipping any spaces) in the formatted data file until another quotation mark is hit. At that point, a space is added to the end of that value and the raw data file is read for another value. In the case that a newline character (\n) is found, it is added to the formatted data file as well to organize the data. Once all values in the raw data have been read, both the raw data file and formatted data file are closed and the function ends.

### checking(Purple)

The checking function is used to compare two strings to determine if they are equal to each other or not. To execute this code two arrays containing strings need to first be passed to the function when it is called. The function then starts with a for loop. The loop starts at 0 and continues for the length of the compare array found by using the strlen() function. Inside the for loop an if statement is used to compare each individual char value in both of the arrays against each other. If even one char value inside of the arrays are not equal to each other, 1 is returned to where the checking function was initially called. If all the char values inside the arrays are equal to each other, when the loop ends and the code exits the loop, 0 is returned to where the checking function was initially called. 1 returning means the strings are not equal to each other and a 0 returning means the arrays are equal to each other.

### columns(Magenta)

This function is used to allocate each individual column in the file a number. Knowing the number assigned to each column allows for the rest of the code to run smoother as the code is already aware of which column is needed when it wishes to extract data. The execution of this code required array's to be initialized with each of the required headings (GEO, VALUE, REF_DATE, Age group, Sex). The code then proceeds to enter a while loop that will run until numbers for all 5 of the required headings have been found and recorded/saved in a global variable. Upon entering the while loop the first heading is scanned from the file and compared against a list of if and else if statements aiming at finding a match to the 5 original headings. In order to find this match the if and else if statements enter the checking function mentioned above. If a match is found the column number is saved and the while loop repeats until the rest of the 4 column headings have been found. If a match is not found the while loop still repeats until all 5 headings have been located.

## Calculations

### Q1.

### 1a. ( Explanation paragraph )

This problem consists of finding the Provincial percent averages for Ontario, Quebec, British Columbia, and Alberta. To solve this problem each of the provinces are assigned an array. The code then enters a while loop that will run until the end of the file. At the start of the while loop a for loop is used to get to the GEO column containing the province names. The code knows where the column is due to the column function mentioned above. Once the code is at column GEO the province is saved in an array called check. Then proceeding through the code the array check is

compared against a list of if and else if statements looking for a match between the province saved in check against the original province names saved in their own individual arrays. To find this match the if or else if statement calls upon the checking function mentioned above. Once the code knows which if and/or else statement the province in check is associated with it proceeds to reading the rest of the row in the file until reaching the VALUE column. Once at the VALUE column the number in the form of a string is saved in an array called value. While using the function atof() the string is then converted to a double and saved in the variable called num. If num is not equal to 0 the code then enters another if statement where it adds the number to the provincial average as well as a 1 to the provinces count. Num equaling 0 means that the code came across a blank and it needs to be ignored. This process is repeated until the code reaches the file's end. Upon exiting the while loop the code then proceeds to calculate the average for each of the provinces. This average is calculated by taking the sum of all the values saved in the province's average variable and dividing that by its count. At the end the averages along with their province is printed as an output.

## 1a. ( Output screenshot )

```
Question 1 a

The average for Ontario is: 11.70 %
The average for Quebec is: 10.45 %
The average for British Columbia is: 9.72 %
The average for Alberta is: 10.86 %
```

## 1b. ( Explanation paragraph )

Question 1(b) asks for a national average, excluding territories, for all years and age groups. This question is answered in the function titled, “question1b”. The function begins by opening the file, “formattedData.txt”, and using a while loop to continue reading from the formatted data file until it reaches the end of the file. A for loop is then used in order to obtain information from the “GEO” column of the file and store it in an array called, “check”. Canada, excluding territories, is also stored in a separate array which is then compared to “check”. This allows the code to read the appropriate number from the ‘VALUE” column for each line and store that number as a string in another array, “value”. This number is then converted into a double by using the atof() function and stored into a variable called “num”. This allows calculations to be made with the numbers obtained from the code. The sum is calculated and averaged out accordingly by dividing the sum by the amount of values obtained, which is stored in a counter.

## 1b. ( Output screenshot )

```
Question 1 b

The average for Canada excluding territories is: 10.87 %
```

## 1c. ( Explanation paragraph )

For question 1c, we are asked to find the yearly diabetes percentage averages of each province and the whole country from 2015 to 2021 (for a total of 35 averages calculated). In the program, the function that performs these calculations starts with opening the formatted data file that has the values needed to determine these averages. With the file open, a “while” loop is created that takes in each line of the formatted data until there are no lines left. In this loop, the first value of the line is stored and another “while” loop is executed to check if this same value is from a column of values needed to solve the problem. If it is not, then the next value of the line is stored and checked. If it is a needed value, then it is stored in a variable for later use. Once all of the needed values for this line are stored, they are used in a series of “if” and “for” statements to find where the diabetes average value from this line needs to be sorted. Once found, this value is added to the designated average. Designated counts (stored in arrays) are then used to keep track of how many values are added to these designated averages. In the case that a proper value is not found for the line, the line is skipped entirely from the calculation. Once all the percentage values have been stored, the final averages are calculated (by dividing the designated average with their designated count) and displayed on the console at the end of the function. This data is also stored in a “.dat” file that will be used in question 5.

## 1c. ( Output screenshot )

```
Question 1 c

2015 Average for Canada(excludingterritories): 10.60%
2015 Average for Ontario: 10.77%
2015 Average for Alberta: 9.32%
2015 Average for Quebec: 10.90%
2015 Average for BritishColumbia: 9.30%

2016 Average for Canada(excludingterritories): 10.70%
2016 Average for Ontario: 12.20%
2016 Average for Alberta: 9.77%
2016 Average for Quebec: 9.82%
2016 Average for BritishColumbia: 8.53%

2017 Average for Canada(excludingterritories): 10.95%
2017 Average for Ontario: 11.98%
2017 Average for Alberta: 11.97%
2017 Average for Quebec: 9.58%
2017 Average for BritishColumbia: 10.14%

2018 Average for Canada(excludingterritories): 10.78%
2018 Average for Ontario: 11.28%
2018 Average for Alberta: 11.02%
2018 Average for Quebec: 10.65%
2018 Average for BritishColumbia: 8.52%

2019 Average for Canada(excludingterritories): 11.70%
2019 Average for Ontario: 13.03%
2019 Average for Alberta: 11.33%
2019 Average for Quebec: 10.48%
2019 Average for BritishColumbia: 11.44%

2020 Average for Canada(excludingterritories): 10.60%
2020 Average for Ontario: 11.17%
2020 Average for Alberta: 12.88%
2020 Average for Quebec: 11.42%
2020 Average for BritishColumbia: 9.04%

2021 Average for Canada(excludingterritories): 10.75%
2021 Average for Ontario: 11.48%
2021 Average for Alberta: 9.82%
2021 Average for Quebec: 10.47%
2021 Average for BritishColumbia: 11.65%
```

## 1d ( Explanation paragraph )

In this problem the averages for each of the provinces plus Canada (excluding territories) need to be found according to the age groups 35-49, 50-64, and 65+. In order to start this program arrays need to be assigned to the provinces, Canada (excluding territories), as well as each of the individual age groups. The program then enters a while loop that will run until the end of the file. Inside the while loop the code will run until reaching the GEO column. At the GEO column the province or the country's name will be saved in an array called check. The code will then check to see which if or else if statement the string in check corresponds to. Upon entering the statement with the corresponding name the code will then proceed to read the row it is on until the Age group and the VALUE column. At each column it will save the data stored there in an array called age and value respectively. The number in the form of a string saved in the value array will then be converted into a double using the atof() function and will be saved in the variable num. If num is not zero the code will enter an if statement. If num is zero the code will ignore it and continue since a zero num equals a blank space. Entering the if statement the program will run into another list of if and else if statement used to compare the ages. These statements are looking for which age the value is associated with and upon finding its corresponding age it will add the number in num to the provinces or countries average variable as well as and a one to its count. This process will continue until the end of the file. When the code reaches the files end the program will exit the while loop and will proceed to calculate the averages. The averages are calculated by taking the sum of the values saved in the provinces or the countries variable and dividing it by its count. The program will then print out the averages as an output.

## 1d. ( Output screenshot )

```
Question 1 d

The average for Ontario is:
35 to 49 years: 4.64%
50 to 64 years: 11.22%
65 years and over: 19.24%

The average for Quebec is:
35 to 49 years: 3.35%
50 to 64 years: 9.06%
65 years and over: 18.44%

The average for British Columbia is:
35 to 49 years: 3.43%
50 to 64 years: 7.91%
65 years and over: 15.18%

The average for Alberta is:
35 to 49 years: 4.46%
50 to 64 years: 10.29%
65 years and over: 16.92%

The average for Canada (excluding territories) is:
35 to 49 years: 4.06%
50 to 64 years: 10.33%
65 years and over: 18.21%
```

## Q2. ( Explanation paragraph )

This problem is connected to question 1(a). The problem consists of ordering the provincial averages found in question 1(a) to find the highest and lowest provincial diabetes average. The program starts with assigning each of the province averages to an index in one array called prov_avg. There are then two variables called comp_high and comp_low. These variables start with Ontario's average and are used to compare the rest of the averages against them. The last two variables needed are called high and low. These two variables will hold the location of the highest and lowest average province. They will start with 0 indicating the location of Ontario. The program then enters a for loop and runs for a total of 4 times as there are only 4 provinces. The if and else if statement in the for loop compares each of the averages inside the array prov_avg with the comparing variable to find an average that is greater than or less than the average saved in the compare variable. If a new average in the prov_avg variable is found to have a greater then or less than average the program enters the statement and saves the new location of the average indicating a different place as well can changes the comp_high or comp_low variable to equal the new average needing to be compared. Upon exiting the for loop the average saved in the compare variables should be the highest and lowest average and their associating locations should have also been saved. The program will then go through a series of switch case statements and will enter the statement carrying the number associated with the location of the highest or lowest average. It will then print the highest and lowest province along with their average as an output.

## Q2. ( Output screenshot )

```
Question 2

The highest Provincial average percentage for diabetes belongs to...
Ontario @ 11.70 %

The lowest Provincial average percentage for diabetes belongs to...
British Columbia @ 9.72 %
```

## Q3. ( Explanation paragraph )

Question 3 asks to indicate which individual provinces have a higher diabetes average than the national average, and which individual provinces have a lower diabetes average lower than the national average. This question is answered in the function titled, “question3”. Since each of the provincial averages along with the national average were all stored as global variables, this function is able to call on each average previously calculated and compare each provincial average with the national average. By using a series of if statements, it was determined that Ontario has a higher percentage average of diabetes than the national average, while the rest of the provinces each have a lower percentage average than the national average.

Q3. ( Output screenshot )

```
Question 3

Ontario is above the national average
Quebec is below the national average
British Columbia is below the national average
Alberta is below the national average
```

Q4. ( Explanation paragraph )

Question 4 asks us to indicate which years and provinces have the highest and lowest percentage of diabetes. In the program, this is accomplished through the “question4()” function with the data from question 1c that tells us all of the average diabetes percentage averages of each province for each year. This function uses a basic “for” statement that goes through each average from the question 1c data and compares it with the minimum and maximum averages stored in the function. If it finds an average smaller than the minimum average, that average becomes the new minimum average. If it finds an average bigger than the maximum average, that average becomes the new maximum average. When one of these two cases occur, the province and year this occurs is also stored for later use. This “for” statement is repeated for each year in the given data. Once all the given data is gone through, the results are displayed on the console at the end of the function.

Q4. ( Output screenshot )

```
Question 4

Min Average...
Year: 2018
Place: BritishColumbia
Average: 8.52%

Max Average...
Year: 2019
Place: Ontario
Average: 13.03%
```

## Gnuplot Questions

Q5. ( Explanation paragraph )

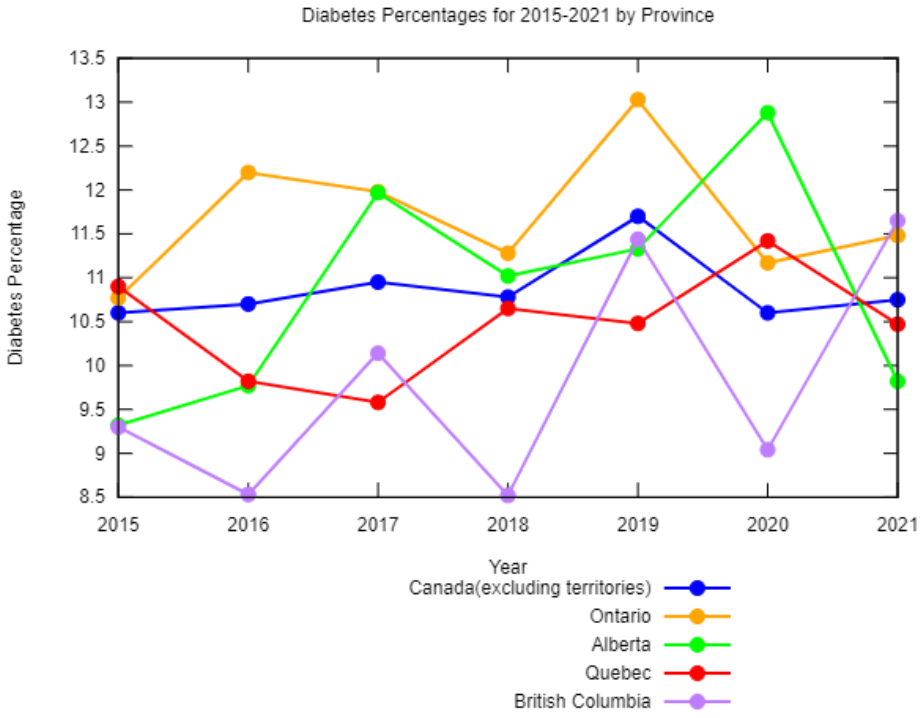
To create a graph of the diabetes percentages for the years 2015 to 2021 (all age groups and genders together) for the four provinces and the national average (indicated as Canada excluding territories), we can use GNUPlot. First, we need to extract the relevant data from the CSV file using C programming, then create a file called “q5plot.dat” within our code and write the appreciative data (data collected in q1c) inside it, download the file and open it using notepad

and save it as q5plot.dat. Then write a script in Gnuplot that will generate the graph. We can use the "plot" command in Gnuplot to plot the data for each province and the national average on the same graph. We can use different line styles and/or colors for each line plot to distinguish them. We also need to label the axes clearly and add a title to the graph. Finally, we should include a legend that shows the meaning of the five lines. The resulting graph will allow us to visualize how the prevalence of diabetes varies over time in each province and in Canada as a whole.


File created within our code, named “q5plot.dat”:

# Year Canada(excludingterritories) Ontario Alberta Quebec BritishColumbia
2015 10.60 10.77 9.32 10.90 9.30
2016 10.70 12.20 9.77 9.82 8.53
2017 10.95 11.98 11.97 9.58 10.14
2018 10.78 11.28 11.02 10.65 8.52
2019 11.70 13.03 11.33 10.48 11.44
2020 10.60 11.17 12.88 11.42 9.04
2021 10.75 11.48 9.82 10.47 11.65

## Q5. ( PNG file (plot image) )



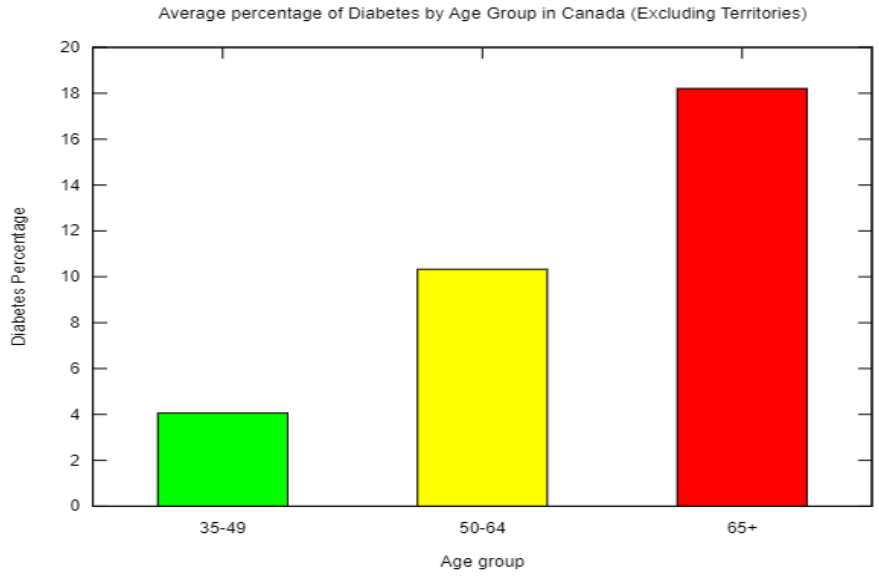
## Q6. ( Explanation paragraph )

To create a bar chart in GNUPlot showing the average percentages of diabetes among the three age groups for Canada (excluding territories), the following steps were taken. The relevant data was extracted from the CSV file using C programming, and a file called "q6plot.dat" was created and filled with the data (data collected in q1d). The file was then downloaded and opened using notepad, then saved as q6plot.dat. The GNUPlot script was then written, starting with setting the title and axis labels. The bar style was set to be filled with solid colors, and the width and style of the boxes were defined. The data style was set to histogram, with bars clustered and a gap of 1 between them. The fill style of the histogram was set to be solid with no border. Linetypes were defined for each bar with a different color using RGB, and the y-axis range was set to start from 0 and go to an automatic upper bound. The data were plotted using the using keyword to specify columns for the x and y values and the color index. The bars were defined to be boxes, and their color was determined by the linetype. Ultimately, the plot shows 3 bars, each with a different color representing the Diabetes percentage for their specific age group.


File created within our code, named “q6plot.dat”:

```
# age   Canada(excluding territories)
35-49   4.06
50-64   10.33
65+     18.21
```

## Q6. ( PNG file (plot image) )



## Conclusion

In conclusion, this project provided an opportunity to work with real data collected by Statistics Canada and apply C programming and GNUPlot functionalities to perform calculations, create tables, and generate graphs. It allowed us to gain practical experience in data analysis and visualization and apply the skills learned in class to a real-world problem.

Overall, the project was challenging, but it was a valuable learning experience. The process of reading the data from the CSV file, converting it to numerical values, and performing the required calculations required a lot of attention to detail and patience. We learned the importance of keeping the code organized and well-documented to facilitate the process of debugging and troubleshooting.

If we were to do this project again, we would ensure to allocate more time for planning and organizing the code. Additionally, we would explore different methods for data visualization to improve the presentation of our results.

In conclusion, we are grateful for the opportunity to work on this project, and we are confident that the skills and knowledge gained from this experience will be useful in our future projects.

## Gnuplot script for questions 5 and 6

### Q5 Gnuplot script:

```
#set the title and axis labels
set title 'Diabetes Percentages for 2015-2021 by Province'
set xlabel 'Year'
set ylabel 'Diabetes Percentage'

# Define the line styles and colors for each region
set style line 1 lc rgb 'blue' lw 2 pt 7 ps 1 # Canada(excluding territories)
set style line 2 lc rgb 'orange' lw 2 pt 7 ps 1 # Ontario
set style line 3 lc rgb 'green' lw 2 pt 7 ps 1 # Alberta
set style line 4 lc rgb 'red' lw 2 pt 7 ps 1 # Quebec
set style line 5 lc rgb 'purple' lw 2 pt 7 ps 1 # British Columbia

# Set the x-axis range and format
set xrange [2015:2021]
set xtics 2015, 1, 2021
set format x '%4.0f'

#set the position of the legends
set key horizontal center bottom outside
```

```
# Plot the data as line plots with points at the second column values
plot 'q5plot.dat' using 1:2 with linespoints linestyle 1 title
'Canada(excluding territories)', \
     'q5plot.dat' using 1:3 with linespoints linestyle 2 title 'Ontario', \
     'q5plot.dat' using 1:4 with linespoints linestyle 3 title 'Alberta', \
     'q5plot.dat' using 1:5 with linespoints linestyle 4 title 'Quebec', \
     'q5plot.dat' using 1:6 with linespoints linestyle 5 title 'British
Columbia'
```

## Q6 Gnuplot script:

```
#set the title and axis labels
set title 'Average percentage of Diabetes by Age Group in Canada (Excluding
Territories)'
set xlabel 'Age group'
set ylabel 'Diabetes Percentage'

#set the style of the plots
set style fill solid
set boxwidth 0.5
set style data histogram
set style histogram cluster gap 1
set style fill solid border -1

#set the line types for each bar
set linetype 1 lc rgb 'green'
set linetype 2 lc rgb 'yellow'
set linetype 3 lc rgb 'red'

#set the y-axis range to start from 0 and leave the upper bound automatic
set yrange [0:*]

unset key

# Plot the data as a bar chart with variable colors based on line types
# The 'using' keyword specifies the columns to use for the x and y values, as
well as the color index
# The '0:2' tells gnuplot to use the first column (x) and second column (y) of
the data file
# The '($0+1)' adds 1 to the x values to center the bars on the x-axis ticks
# The 'xtic(1)' sets the x-axis labels to the values in the first column
plot 'q6plot.dat' using 0:2:($0+1):xtic(1) with boxes  linecolor variable
```

## Complete C code for Questions 1-4

```
// C libraries to be used in the code
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Global variables to be used in the code */

// Column numbers to where each column is in the formatted data
int geo_count, val_count, age_count, date_count, gender_count;
double ont_avg, que_avg, brit_avg, alb_avg; // Provincial Averages
double nationalavg; // Canada(excludingterritories) Average
char names[5][50] = {"Canada(excludingterritories)", "Ontario", "Alberta",
                     "Quebec", "BritishColumbia"};
// Arrays that store each places' averages depending on their place in the
// names[][] array
double avg_2015[5] = {0, 0, 0, 0, 0}, avg_2016[5] = {0, 0, 0, 0, 0},
       avg_2017[5] = {0, 0, 0, 0, 0}, avg_2018[5] = {0, 0, 0, 0, 0};
double avg_2019[5] = {0, 0, 0, 0, 0}, avg_2020[5] = {0, 0, 0, 0, 0},
       avg_2021[5] = {0, 0, 0, 0, 0};
// Averages for each Province depending on their age group
double ont_35_avg = 0, ont_50_avg = 0, ont_65_avg = 0, que_35_avg = 0,
       que_50_avg = 0, que_65_avg = 0, brit_35_avg = 0, brit_50_avg = 0,
       brit_65_avg = 0, alb_35_avg = 0, alb_50_avg = 0, alb_65_avg = 0,
       can_35_avg = 0, can_50_avg = 0, can_65_avg = 0;

/* converts csv file into a txt file */
void format() {
  // Files and variables to be used in the function
  FILE *in = fopen("statscan_diabetes.csv", "r");
  FILE *formattedData = fopen("formattedData.txt", "w");
  char input;
  int count = 0;
  // While function that goes through the given .csv file
  while (!feof(in)) {
    // Takes in each character of the given file
    fscanf(in, "%c", &input);
    // When a cell value from the .csv file is found, it is formatted and sent
    // to the formatted data file
    if (input == '"') {
      count++;
      while (count == 1) {
        fscanf(in, "%c", &input);
        if (input == '"') {
          fprintf(formattedData, " ");
          count--;
        } else {
```

```
          if (input != ' ')
            fprintf(formattedData, "%c", input);
        }
      }
      // If a newline character is found, the formatted data file also creates
a
      // new line
    } else if (input == '\n') {
      fprintf(formattedData, "\n");
    }
  }

  // Closes the files used in the code
  fclose(in);
  fclose(formattedData);
}

/* Checks if the file string equals value needed */
int checking(char comp[], char org[]) {
  for (int i = 0; i < strlen(comp); i++) {
    if (comp[i] != org[i]) {
      return (1);
      // if not equal to needed string
    }
  }

  return (0); // if equal to needed string
}

/* assigns columns needed with a number */
void columns() {
  // Opens the files and variables to be used in the function
  FILE *formattedData = fopen("formattedData.txt", "r");
  int count = 1, columnsFound = 0;
  char check[50];

  // column headers needing to be found (given in assignment)
  char geo[] = {'G', 'E', 'O'}, val[] = {'V', 'A', 'L', 'U', 'E'},
       date[] = {'R', 'E', 'F', '_', 'D', 'A', 'T', 'E'},
       age[] = {'A', 'g', 'e', 'g', 'r', 'o', 'u', 'p'},
       gender[] = {'S', 'e', 'x'};

  // Find where each column is in the formatted data until all are found
  while (columnsFound != 5) {
    fscanf(formattedData, "%s", check);
    if (checking(check, geo) == 0) {
      geo_count = count;
      columnsFound++;
    } else if (checking(check, val) == 0) {
```

```
      val_count = count;
      columnsFound++;
    } else if (checking(check, age) == 0) {
      age_count = count;
      columnsFound++;
    } else if (checking(check, date) == 0) {
      date_count = count;
      columnsFound++;
    } else if (checking(check, gender) == 0) {
      gender_count = count;
      columnsFound++;
    }
    count++;
  }

  // Closes the formatted data file
  fclose(formattedData);
}

/* Question 1 a */
void question1a() {
  FILE *formattedData = fopen("formattedData.txt", "r");

  char check[50], value[50];
  double num;
  int ont_count = 0, que_count = 0, brit_count = 0, alb_count = 0;

  // rows needing to find
  char ont[] = {'O', 'n', 't', 'a', 'r', 'i', 'o'},
       que[] = {'Q', 'u', 'e', 'b', 'e', 'c'},
       brit[] = {'B', 'r', 'i', 't', 'i', 's', 'h', 'C',
                 'o', 'l', 'u', 'm', 'b', 'i', 'a'},
       alb[] = {'A', 'l', 'b', 'e', 'r', 't', 'a'};

  while (!feof(formattedData)) {
    for (int i = 1; i <= geo_count; i++) {
      fscanf(formattedData, "%s", check);
    }

    if (checking(check, ont) == 0) {
      for (int i = geo_count + 1; i <= val_count; i++) {
        fscanf(formattedData, "%s", value);
      }

      num = atof(value); // converts string to double

      if (num != 0) {
        ont_avg += num; // adds num to sum
        ont_count++;    // adds to count
```

```
    }
  } else if (checking(check, que) == 0) {
    for (int i = geo_count + 1; i <= val_count; i++) {
      fscanf(formattedData, "%s", value);
    }

    num = atof(value);

    if (num != 0) {
      que_avg += num;
      que_count++;
    }
  } else if (checking(check, brit) == 0) {
    for (int i = geo_count + 1; i <= val_count; i++) {
      fscanf(formattedData, "%s", value);
    }
    num = atof(value);

    if (num != 0) {
      brit_avg += num;
      brit_count++;
    }
  } else if (checking(check, alb) == 0) {
    for (int i = geo_count + 1; i <= val_count; i++) {
      fscanf(formattedData, "%s", value);
    }

    num = atof(value);

    if (num != 0) {
      alb_avg += num;
      alb_count++;
    }
  }

  // skips the rest of the row
  fscanf(formattedData, "%*[^\n]");
}

// calculates the average
ont_avg = ont_avg / ont_count;
que_avg = que_avg / que_count;
brit_avg = brit_avg / brit_count;
alb_avg = alb_avg / alb_count;

printf("\nQuestion 1 a\n");
printf("\nThe average for Ontario is: %.2lf %%", ont_avg);
printf("\nThe average for Quebec is: %.2lf %%", que_avg);
printf("\nThe average for British Columbia is: %.2lf %%", brit_avg);
```

```
  printf("\nThe average for Alberta is: %.2lf %%", alb_avg);

  fclose(formattedData);
}

/* Question 1 b */
void question1b() {
  FILE *formattedData = fopen("formattedData.txt", "r");

  char check[50], value[50];
  double num;
  double nationaltotal = 0;
  int nationalcount = 0;

  // rows needing to find
  char nat[] = {'C', 'a', 'n', 'a', 'd', 'a', '(', 'e', 'x', 'c',
                'l', 'u', 'd', 'i', 'n', 'g', 't', 'e', 'r', 'r',
                'i', 't', 'o', 'r', 'i', 'e', 's', ')'};

  while (!feof(formattedData)) {
    for (int i = 1; i <= geo_count; i++) {
      fscanf(formattedData, "%s", check);
    }

    if (checking(check, nat) == 0) {
      for (int i = geo_count + 2; i <= val_count; i++) {
        fscanf(formattedData, "%s", value);
      }

      num = atof(value);

      if (num != 0) {
        nationaltotal += num;
        nationalcount++;
      }
    }

    // skips the rest of the row
    fscanf(formattedData, "%*[^\n]");
  }

  nationalavg = nationaltotal / nationalcount;
  printf("\n\nQuestion 1 b\n");
  printf("\nThe average for Canada excluding territories is: %.2lf %%",
         nationalavg);
  fclose(formattedData);
}

/* Question 1 c */
```

```
void question1c() {
  // File with formatted data to be used in code
  FILE *formattedData = fopen("formattedData.txt", "r");
  // Variables and arrays to be used in the code
  char line[1024], value[50], place[50], date[50], avg[50];
  char *pointer;
  int count = 1, trigger = 0;
  int count2015[5] = {0, 0, 0, 0, 0}, count2016[5] = {0, 0, 0, 0, 0},
      count2017[5] = {0, 0, 0, 0, 0}, count2018[5] = {0, 0, 0, 0, 0};
  int count2019[5] = {0, 0, 0, 0, 0}, count2020[5] = {0, 0, 0, 0, 0},
      count2021[5] = {0, 0, 0, 0, 0};
  /* Date_count , geo_count, and val_count come from columns() function
   * They keep track of which column a required value is from
   */

  // While statement that goes through each line in the formatted code
  while (fgets(&line[0], 1024, formattedData) != NULL) {
    // Pointer variable that stores the column of the line
    pointer = strtok(line, " ");
    // While statement that goes through each column value in the line
    while (pointer != NULL) {
      // Variable that stores each column value in the line
      sscanf(pointer, "%s", value);
      // If statements that store the data needed from the line
      // Count variable keeps track on which column the pointer is at
      // in the formatted data line
      if (count == date_count) {
        strcpy(date, value);
      } else if (count == geo_count &&
                 strcmp(value, "Canada(excludingterritories)") == 0) {
        strcpy(place, value);
        // Since a line with this place value skews the column count,
        // an extra count is added to account for that
        count++;
      } else if (count == geo_count) {
        strcpy(place, value);
      } else if (count == val_count) {
        // If the average value is found to not be a number,
        // Trigger value is raised for the line
        trigger = 0;
        for (int i = 0; i < strlen(value); i++) {
          if (!isdigit(value[i]) && value[i] != '.') {
            trigger = 1;
          }
        }
        strcpy(avg, value);
      }
      // Advances to next column in the line
      count++;
```

```
  pointer = strtok(NULL, " ");
}
// When line reading is done, count is reset
count = 1;
// Sorts the data found depending on year of data
// If trigger was raised, the data for the line is not used in final
// result
if (strcmp(date, "2015") == 0 && trigger == 0) {
  for (int i = 0; i < 5; i++) {
    // Goes through each place to find where to store it
    if (strcmp(place, names[i]) == 0) {
      avg_2015[i] += atof(avg);
      // Counts how many values were added to average for later use
      count2015[i]++;
    }
  }
}
// Same process from 2015 is done for the rest of the years
if (strcmp(date, "2016") == 0 && trigger == 0) {
  for (int i = 0; i < 5; i++) {
    if (strcmp(place, names[i]) == 0) {
      avg_2016[i] += atof(avg);
      count2016[i]++;
    }
  }
}
if (strcmp(date, "2017") == 0 && trigger == 0) {
  for (int i = 0; i < 5; i++) {
    if (strcmp(place, names[i]) == 0) {
      avg_2017[i] = atof(avg) + avg_2017[i];
      count2017[i]++;
    }
  }
}
if (strcmp(date, "2018") == 0 && trigger == 0) {
  for (int i = 0; i < 5; i++) {
    if (strcmp(place, names[i]) == 0) {
      avg_2018[i] += atof(avg);
      count2018[i]++;
    }
  }
}
if (strcmp(date, "2019") == 0 && trigger == 0) {
  for (int i = 0; i < 5; i++) {
    if (strcmp(place, names[i]) == 0) {
      avg_2019[i] += atof(avg);
      count2019[i]++;
    }
  }
```

```
  }
  if (strcmp(date, "2020") == 0 && trigger == 0) {
    for (int i = 0; i < 5; i++) {
      if (strcmp(place, names[i]) == 0) {
        avg_2020[i] += atof(avg);
        count2020[i]++;
      }
    }
  }
  if (strcmp(date, "2021") == 0 && trigger == 0) {
    for (int i = 0; i < 5; i++) {
      if (strcmp(place, names[i]) == 0) {
        avg_2021[i] += atof(avg);
        count2021[i]++;
      }
    }
  }
}
// Prints the header for the results to the question
printf("\n\nQuestion 1 c\n\n");

// Opens a file to store all the data needed to answer Question 5
FILE *q5 = fopen("q5plot.dat", "w");
// Prints the heading line needed in the new file before storing data for
/* Question 5 */
fprintf(q5, "# Year Canada(excludingterritories) Ontario Alberta Quebec "
            "BritishColumbia\n");
// For statement that calculates the average of each year from each place
for (int i = 0; i < 5; i++) {
  avg_2015[i] /= count2015[i];
  // Prints the results of each average of each year from each place
  printf("2015 Average for %s: %.2lf%%\n", names[i], avg_2015[i]);
}
printf("\n");
// Stores the data of each average from each place for the each specified
fprintf(q5, "2015  %.2lf  %.2lf  %.2lf  %.2lf  %.2lf\n", avg_2015[0],
        avg_2015[1], avg_2015[2], avg_2015[3], avg_2015[4]);
// Same process done for the rest of the years checked
for (int i = 0; i < 5; i++) {
  avg_2016[i] /= count2016[i];
  printf("2016 Average for %s: %.2lf%%\n", names[i], avg_2016[i]);
}
printf("\n");
fprintf(q5, "2016  %.2lf  %.2lf  %.2lf  %.2lf  %.2lf\n", avg_2016[0],
        avg_2016[1], avg_2016[2], avg_2016[3], avg_2016[4]);
for (int i = 0; i < 5; i++) {
  avg_2017[i] /= count2017[i];
  printf("2017 Average for %s: %.2lf%%\n", names[i], avg_2017[i]);
}
```

```
  printf("\n");
  fprintf(q5, "2017  %.2lf  %.2lf  %.2lf  %.2lf  %.2lf\n", avg_2017[0],
          avg_2017[1], avg_2017[2], avg_2017[3], avg_2017[4]);
  for (int i = 0; i < 5; i++) {
    avg_2018[i] /= count2018[i];
    printf("2018 Average for %s: %.2lf%%\n", names[i], avg_2018[i]);
  }
  printf("\n");
  fprintf(q5, "2018  %.2lf  %.2lf  %.2lf  %.2lf  %.2lf\n", avg_2018[0],
          avg_2018[1], avg_2018[2], avg_2018[3], avg_2018[4]);
  for (int i = 0; i < 5; i++) {
    avg_2019[i] /= count2019[i];
    printf("2019 Average for %s: %.2lf%%\n", names[i], avg_2019[i]);
  }
  printf("\n");
  fprintf(q5, "2019  %.2lf  %.2lf  %.2lf  %.2lf  %.2lf\n", avg_2019[0],
          avg_2019[1], avg_2019[2], avg_2019[3], avg_2019[4]);
  for (int i = 0; i < 5; i++) {
    avg_2020[i] /= count2020[i];
    printf("2020 Average for %s: %.2lf%%\n", names[i], avg_2020[i]);
  }
  printf("\n");
  fprintf(q5, "2020  %.2lf  %.2lf  %.2lf  %.2lf  %.2lf\n", avg_2020[0],
          avg_2020[1], avg_2020[2], avg_2020[3], avg_2020[4]);
  for (int i = 0; i < 5; i++) {
    avg_2021[i] /= count2021[i];
    printf("2021 Average for %s: %.2lf%%\n", names[i], avg_2021[i]);
  }
  fprintf(q5, "2021  %.2lf  %.2lf  %.2lf  %.2lf  %.2lf\n", avg_2021[0],
          avg_2021[1], avg_2021[2], avg_2021[3], avg_2021[4]);

  // Closes all files used in the function
  fclose(q5);
  fclose(formattedData);
}

/* Question 1 d */
void question1d() {
  FILE *formattedData = fopen("formattedData.txt", "r");

  char check[50], value[50], age[50];
  double num;
  int ont_35_count = 0, ont_50_count = 0, ont_65_count = 0, que_35_count = 0,
      que_50_count = 0, que_65_count = 0, brit_35_count = 0, brit_50_count =
0,
      brit_65_count = 0, alb_35_count = 0, alb_50_count = 0, alb_65_count = 0,
      can_35_count = 0, can_50_count = 0, can_65_count = 0;

  // rows needing to find
```

```
char ont[] = "Ontario", que[] = "Quebec", brit[] = "BritishColumbia",
     alb[] = "Alberta", can[] = "Canada(excludingterritories)";
char age_35[] = "35to49years", age_50[] = "50to64years",
     age_65[] = "65yearsandover";

while (!feof(formattedData)) {
  for (int i = 1; i <= geo_count; i++) {
    fscanf(formattedData, "%s", check);
  }

  if (checking(check, ont) == 0) {
    for (int i = geo_count + 1; i <= age_count; i++) {
      fscanf(formattedData, "%s", age);
    }
    for (int i = age_count + 1; i <= val_count; i++) {
      fscanf(formattedData, "%s", value);
    }

    num = atof(value); // converts string to double

    if (num != 0) {
      if (checking(age, age_35) == 0) {
        ont_35_avg += num; // adds num to sum
        ont_35_count++;    // adds to count
      } else if (checking(age, age_50) == 0) {
        ont_50_avg += num;
        ont_50_count++;
      } else if (checking(age, age_65) == 0) {
        ont_65_avg += num;
        ont_65_count++;
      }
    }
  } else if (checking(check, que) == 0) {
    for (int i = geo_count + 1; i <= age_count; i++) {
      fscanf(formattedData, "%s", age);
    }

    for (int i = age_count + 1; i <= val_count; i++) {
      fscanf(formattedData, "%s", value);
    }

    num = atof(value); // converts string to double

    if (num != 0) {
      if (checking(age, age_35) == 0) {
        que_35_avg += num; // adds num to sum
        que_35_count++;    // adds to count
      } else if (checking(age, age_50) == 0) {
        que_50_avg += num;
```

```
      que_50_count++;
    } else if (checking(age, age_65) == 0) {
      que_65_avg += num;
      que_65_count++;
    }
  }
} else if (checking(check, brit) == 0) {
  for (int i = geo_count + 1; i <= age_count; i++) {
    fscanf(formattedData, "%s", age);
  }

  for (int i = age_count + 1; i <= val_count; i++) {
    fscanf(formattedData, "%s", value);
  }

  num = atof(value); // converts string to double

  if (num != 0) {
    if (checking(age, age_35) == 0) {
      brit_35_avg += num; // adds num to sum
      brit_35_count++;    // adds to count
    } else if (checking(age, age_50) == 0) {
      brit_50_avg += num;
      brit_50_count++;
    } else if (checking(age, age_65) == 0) {
      brit_65_avg += num;
      brit_65_count++;
    }
  }
} else if (checking(check, alb) == 0) {
  for (int i = geo_count + 1; i <= age_count; i++) {
    fscanf(formattedData, "%s", age);
  }

  for (int i = age_count + 1; i <= val_count; i++) {
    fscanf(formattedData, "%s", value);
  }

  num = atof(value); // converts string to double

  if (num != 0) {
    if (checking(age, age_35) == 0) {
      alb_35_avg += num; // adds num to sum
      alb_35_count++;    // adds to count
    } else if (checking(age, age_50) == 0) {
      alb_50_avg += num;
      alb_50_count++;

    } else if (checking(age, age_65) == 0) {
```

```
        alb_65_avg += num;
        alb_65_count++;
      }
    }
  } else if (checking(check, can) == 0) {
    for (int i = geo_count + 2; i <= age_count; i++) {
      fscanf(formattedData, "%s", age);
    }

    for (int i = age_count + 1; i <= val_count; i++) {
      fscanf(formattedData, "%s", value);
    }

    num = atof(value); // converts string to double

    if (num != 0) {
      if (checking(age, age_35) == 0) {
        can_35_avg += num; // adds num to sum
        can_35_count++;    // adds to count
      } else if (checking(age, age_50) == 0) {
        can_50_avg += num;
        can_50_count++;

      } else if (checking(age, age_65) == 0) {
        can_65_avg += num;
        can_65_count++;
      }
    }
  }

  // skips the rest of the row
  fscanf(formattedData, "%*[^\n]");
}

// calculates the average
ont_35_avg = ont_35_avg / ont_35_count;
ont_50_avg = ont_50_avg / ont_50_count;
ont_65_avg = ont_65_avg / ont_65_count;

que_35_avg = que_35_avg / que_35_count;
que_50_avg = que_50_avg / que_50_count;
que_65_avg = que_65_avg / que_65_count;

brit_35_avg = brit_35_avg / brit_35_count;
brit_50_avg = brit_50_avg / brit_50_count;
brit_65_avg = brit_65_avg / brit_65_count;

alb_35_avg = alb_35_avg / alb_35_count;
alb_50_avg = alb_50_avg / alb_50_count;
```

```
  alb_65_avg = alb_65_avg / alb_65_count;

  can_35_avg = can_35_avg / can_35_count;
  can_50_avg = can_50_avg / can_50_count;
  can_65_avg = can_65_avg / can_65_count;

  printf("\nQuestion 1 d\n");
  printf("\nThe average for Ontario is:");
  printf("\n35 to 49 years: %.2lf%%", ont_35_avg);
  printf("\n50 to 64 years: %.2lf%%", ont_50_avg);
  printf("\n65 years and over: %.2lf%%\n", ont_65_avg);
  printf("\nThe average for Quebec is:");
  printf("\n35 to 49 years: %.2lf%%", que_35_avg);
  printf("\n50 to 64 years: %.2lf%%", que_50_avg);
  printf("\n65 years and over: %.2lf%%\n", que_65_avg);
  printf("\nThe average for British Columbia is:");
  printf("\n35 to 49 years: %.2lf%%", brit_35_avg);
  printf("\n50 to 64 years: %.2lf%%", brit_50_avg);
  printf("\n65 years and over: %.2lf%%\n", brit_65_avg);
  printf("\nThe average for Alberta is:");
  printf("\n35 to 49 years: %.2lf%%", alb_35_avg);
  printf("\n50 to 64 years: %.2lf%%", alb_50_avg);
  printf("\n65 years and over: %.2lf%%\n", alb_65_avg);
  printf("\nThe average for Canada (excluding territories) is:");
  printf("\n35 to 49 years: %.2lf%%", can_35_avg);
  printf("\n50 to 64 years: %.2lf%%", can_50_avg);
  printf("\n65 years and over: %.2lf%%\n", can_65_avg);

  /* Question 6 */
  FILE *q6 = fopen("q6plot.dat", "w");

  fprintf(q6, "# age\tCanada(excluding territories)\n");
  fprintf(q6, "35-49\t%.2lf\n", can_35_avg);
  fprintf(q6, "50-64\t%.2lf\n", can_50_avg);
  fprintf(q6, "65+\t%.2lf\n", can_65_avg);

  fclose(q6);
  fclose(formattedData);
}

/* Question 2 */
void question2() {
  double prov_avg[4];
  prov_avg[0] = ont_avg;
  prov_avg[1] = que_avg;
  prov_avg[2] = brit_avg;
  prov_avg[3] = alb_avg;

  double comp_low = ont_avg;
```

```
double comp_high = ont_avg;

int low = 0, high = 0;

// getting the greatest and the lowest provincial average
for (int i = 0; i < 4; i++) {
  if (prov_avg[i] < comp_low) {
    comp_low = prov_avg[i];
    low = i;
  } else if (prov_avg[i] > comp_high) {
    comp_high = prov_avg[i];
    high = i;
  }
}

printf("\nQuestion 2\n");

switch (high) {
case 0:
  printf("\nThe highest Provincial average percentage for diabetes belongs "
         "to...\nOntario @ %.2lf %%\n",
         ont_avg);
  break;
case 1:
  printf("\nThe highest Provincial average percentage for diabetes belongs "
         "to...\nQuebec @ %.2lf %%\n",
         que_avg);
  break;
case 2:
  printf("\nThe highest Provincial average percentage for diabetes belongs "
         "to...\nBritish Columbia @ %.2lf %%\n",
         brit_avg);
  break;
case 3:
  printf("\nThe highest Provincial average percentage for diabetes belongs "
         "to...\nAlberta @ %.2lf %%\n",
         alb_avg);
  break;
}

switch (low) {
case 0:
  printf("\nThe lowest Provincial average percentage for diabetes belongs "
         "to...\nOntario @ %.2lf %%\n",
         ont_avg);
  break;
case 1:
  printf("\nThe lowest Provincial average percentage for diabetes belongs "
         "to...\nQuebec @ %.2lf %%\n",
```

```
            que_avg);
    break;
  case 2:
    printf("\nThe lowest Provincial average percentage for diabetes belongs "
           "to...\nBritish Columbia @ %.2lf %%\n",
           brit_avg);
    break;
  case 3:
    printf("\nThe lowest Provincial average percentage for diabetes belongs "
           "to...\nAlberta @ %.2lf %%\n",
           alb_avg);
    break;
  }
}

/* Question 3 */
void question3() {

  printf("\nQuestion 3\n");

  if (ont_avg < nationalavg) {
    printf("\nOntario is below the national average");
  }

  if (ont_avg > nationalavg) {
    printf("\nOntario is above the national average");
  }

  if (que_avg < nationalavg) {
    printf("\nQuebec is below the national average");
  }

  if (que_avg > nationalavg) {
    printf("\nQuebec is above the national average");
  }

  if (brit_avg < nationalavg) {
    printf("\nBritish Columbia is below the national average");
  }

  if (brit_avg > nationalavg) {
    printf("\nBritish Columbia is above the national average");
  }

  if (alb_avg < nationalavg) {
    printf("\nAlberta is below the national average");
  }

  if (alb_avg > nationalavg) {
```

```
    printf("\nAlberta is above the national average");
  }
}

/* Question 4 */
void question4() {
  // Variables to be used in the code
  // Since the bounds to the diabetes percentages is from 0-100, the min
  // and max were set beyond them as placeholders for the first values of the
  // code
  double min = 101, max = -1;
  char minYear[5], minPlace[50], maxYear[5], maxPlace[50];
  // For statement that cycles through each of the averages for this year
  // and finds the province with the highest and lowest averages
  for (int i = 1; i < 5; i++) {
    if (min > avg_2015[i]) {
      min = avg_2015[i];
      strcpy(minYear, "2015");
      strcpy(minPlace, names[i]);
    } else if (max < avg_2015[i]) {
      max = avg_2015[i];
      strcpy(maxYear, "2015");
      strcpy(maxPlace, names[i]);
    }
  }
  // Same process is done for the rest of the years checked
  for (int i = 1; i < 5; i++) {
    if (min > avg_2016[i]) {
      min = avg_2016[i];
      strcpy(minYear, "2016");
      strcpy(minPlace, names[i]);
    } else if (max < avg_2016[i]) {
      max = avg_2016[i];
      strcpy(maxYear, "2016");
      strcpy(maxPlace, names[i]);
    }
  }
  for (int i = 1; i < 5; i++) {
    if (min > avg_2017[i]) {
      min = avg_2017[i];
      strcpy(minYear, "2017");
      strcpy(minPlace, names[i]);
    } else if (max < avg_2017[i]) {
      max = avg_2017[i];
      strcpy(maxYear, "2017");
      strcpy(maxPlace, names[i]);
    }
  }
  for (int i = 1; i < 5; i++) {
```

```
    if (min > avg_2018[i]) {
      min = avg_2018[i];
      strcpy(minYear, "2018");
      strcpy(minPlace, names[i]);
    } else if (max < avg_2018[i]) {
      max = avg_2018[i];
      strcpy(maxYear, "2018");
      strcpy(maxPlace, names[i]);
    }
  }
  for (int i = 1; i < 5; i++) {
    if (min > avg_2019[i]) {
      min = avg_2019[i];
      strcpy(minYear, "2019");
      strcpy(minPlace, names[i]);
    } else if (max < avg_2019[i]) {
      max = avg_2019[i];
      strcpy(maxYear, "2019");
      strcpy(maxPlace, names[i]);
    }
  }
  for (int i = 1; i < 5; i++) {
    if (min > avg_2020[i]) {
      min = avg_2020[i];
      strcpy(minYear, "2020");
      strcpy(minPlace, names[i]);
    } else if (max < avg_2020[i]) {
      max = avg_2020[i];
      strcpy(maxYear, "2020");
      strcpy(maxPlace, names[i]);
    }
  }
  for (int i = 1; i < 5; i++) {
    if (min > avg_2021[i]) {
      min = avg_2021[i];
      strcpy(minYear, "2021");
      strcpy(minPlace, names[i]);
    } else if (max < avg_2021[i]) {
      max = avg_2021[i];
      strcpy(maxYear, "2021");
      strcpy(maxPlace, names[i]);
    }
  }
  // Displays header for the results to this question
  printf("\n\nQuestion 4\n");

  // Displays the answer to the question
  printf("\nMin Average...\nYear: %s \nPlace: %s \nAverage: %.2lf%%\n",
minYear,
```

```
          minPlace, min);
  printf("\nMax Average...\nYear: %s \nPlace: %s \nAverage: %.2lf%%\n",
maxYear,
          maxPlace, max);
}

/* Main Function */
int main(void) {
  format();  // formats the cvs file to txt file
  columns(); // gets the column number of needed data

  // Functions that run the code for each problem
  question1a();
  question1b();
  question1c();
  question1d();

  question2();
  question3();
  question4();

  // Ends the program
  return 0;
}
```