# CPS188 - Term Project : Winter 2023

**Ahmed Tabl (Section 11) & Hamza Musaddaq (Section 13)**

**Dr.Sadaf Mustafiz**

**2/04/2023**

## Introduction:

Diabetes is a chronic condition that affects millions of people worldwide, and it is one of the leading causes of morbidity and mortality globally. In Canada, diabetes is a significant health concern, with an estimated 3.5 million people living with the disease. This project aims to analyze the prevalence of diabetes in Canada's four most populated provinces (Ontario, Quebec, British Columbia, and Alberta), using data collected by Statistics Canada between 2015 and 2021.

The project utilizes C programming to read and manipulate the data in the CSV file provided by Statistics Canada. The relevant columns in the file, such as the year, province, age group, sex, and percentage of the population with diabetes, are extracted and analyzed using various computational techniques. The missing data in the file are ignored to ensure accurate results.

The report presents tables and graphs that illustrate the prevalence of diabetes in the four provinces and Canada as a whole. GNUPlot functionalities are used to generate high-quality visualizations that aid in the analysis of the data. The project draws conclusions from the data and provides insights into the prevalence of diabetes in Canada's most populated provinces.

Overall, this project contributes to the understanding of diabetes in Canada and can aid healthcare professionals and policymakers in designing strategies to prevent and manage the disease.

## How the code works:

This program reads data from a CSV file named "statscan_diabetes.csv" and calculates the average number of diabetic patients in each Canadian province from 2015 to 2021. The program uses a two-dimensional array to store the data and performs calculations on the data for each province.

The program first reads the data from the file and stores it in a two-dimensional array named array. The array has 211 rows and 18 columns, with each row representing a patient and each column representing a data field.

The program then calculates the average number of diabetic patients in each province from 2015 to 2021. The data for each province is stored in rows 43 to 210 of the array array. The program iterates over these rows and extracts the relevant data (the number of diabetic patients) from column 14 of each row. The program removes quotation marks from the data and converts it to a double. The program then calculates the average value for each province using a loop.

The program also calculates the average number of diabetic patients in Canada (excluding territories) from 2015 to 2021. The program iterates over rows 1 to 42 of the array array and calculates the average value in the same way as for the provinces.

Finally, the program prints the results to the console. The program first prints the provincial averages, followed by the national average, and then the yearly averages for each province. The yearly averages are calculated using a nested loop that iterates over the years from 2015 to 2021 and calculates the average number of diabetic patients for each year.

## **Precursor:**

File reading -

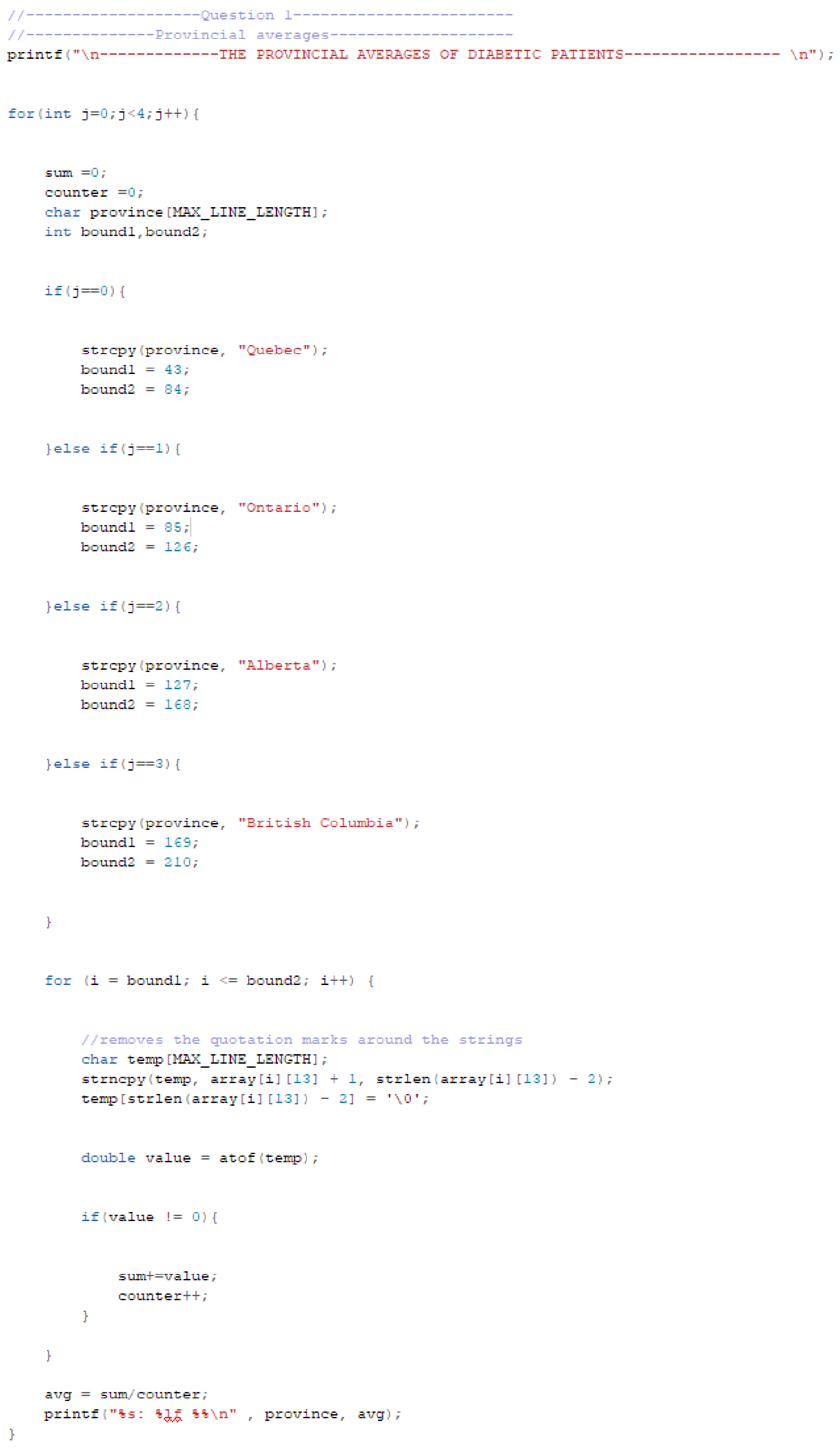
```
file = fopen("statscan_diabetes.csv", "r"); // open file for reading


if (file) {
    while (fgets(line, MAX_LINE_LENGTH, file)) { // read each line of the file
        j = 0;
        token = strtok(line, ","); // split the line into tokens using comma as the delimiter
        while (token != NULL && j < MAX_COLS) { // process each token
            array[i][j] = malloc(strlen(token) + 1); // allocate memory for the token
            strcpy(array[i][j], token); // copy the token to the array
            j++;
            token = strtok(NULL, ","); // get the next token
        }
        i++;
    }


    fclose(file); // close the file
```

The segment of code shown above is responsible for the extraction of the data from the csv file and storing it in a 2D array with the appropriate size. This code reads the csv file line by line, splits each line into tokens using a comma as the delimiter, and stores the tokens in a 2D array of strings. It dynamically allocates memory for each token and copies the token to the array. The maximum length of a line and the maximum number of columns are defined by constants MAX_LINE_LENGTH and MAX_COLS respectively. The code closes the file after reading and processing all lines. In this case, the file contained 211 rows and 18 columns, hence defining the following constants,

```
#define MAX_ROWS 211
#define MAX_COLS 18
```

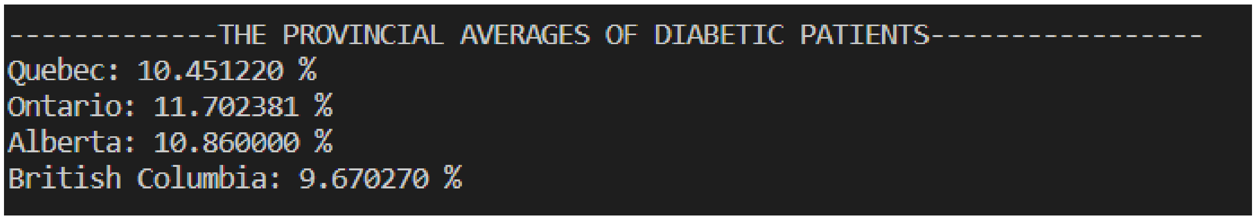
Questions 1 through 4 are all computed via the 2D array values extracted from that code segment.

## Q1) a)



```c
//-------------------Question 1-----------------------
//--------------Provincial averages-------------------
printf("\n------------THE PROVINCIAL AVERAGES OF DIABETIC PATIENTS----------------- \n");


for(int j=0;j<4;j++){


    sum =0;
    counter =0;
    char province[MAX_LINE_LENGTH];
    int bound1,bound2;


    if(j==0){


        strcpy(province, "Quebec");
        bound1 = 43;
        bound2 = 84;


    }else if(j==1){


        strcpy(province, "Ontario");
        bound1 = 85;
        bound2 = 126;


    }else if(j==2){


        strcpy(province, "Alberta");
        bound1 = 127;
        bound2 = 168;


    }else if(j==3){


        strcpy(province, "British Columbia");
        bound1 = 169;
        bound2 = 210;


    }


    for (i = bound1; i <= bound2; i++) {


        //removes the quotation marks around the strings
        char temp[MAX_LINE_LENGTH];
        strncpy(temp, array[i][13] + 1, strlen(array[i][13]) - 2);
        temp[strlen(array[i][13]) - 2] = '\0';


        double value = atof(temp);


        if(value != 0){


            sum+=value;
            counter++;
        }

    }

    avg = sum/counter;
    printf("%s: %lf %%\n" , province, avg);
}
```

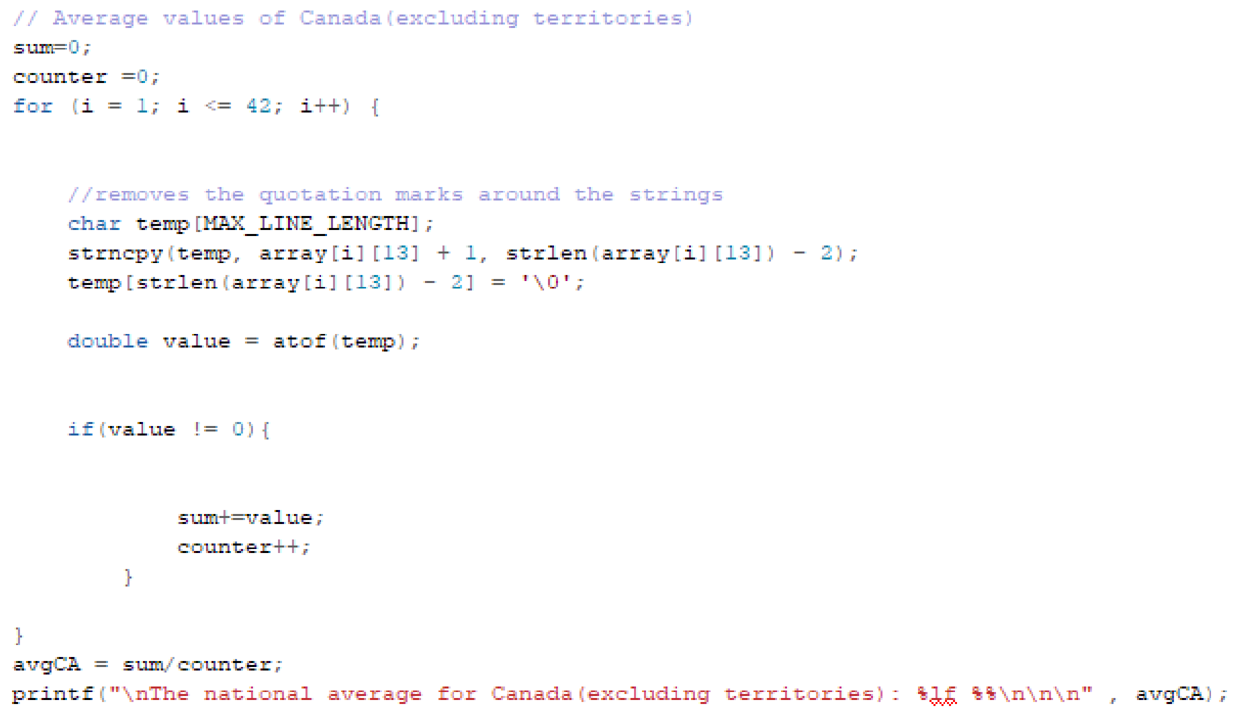
**Output:**



```
-------------THE PROVINCIAL AVERAGES OF DIABETIC PATIENTS-----------------
Quebec: 10.451220 %
Ontario: 11.702381 %
Alberta: 10.860000 %
British Columbia: 9.670270 %
```

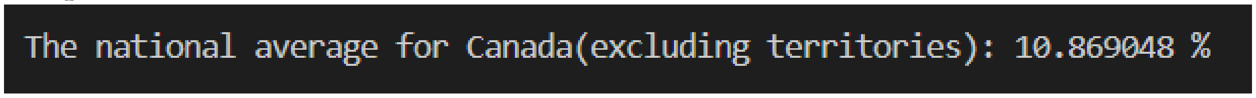
The above code calculates the provincial averages of diabetic patients based on data stored in the 2D array of strings. It first prints a header message, and then iterates over the four provinces, each with a specific range of rows in the array, corresponding to the rows in the csv file. For each province, the code initializes a sum and counter variable to calculate the average, and also defines a province name and row boundaries based on the current iteration. It then loops through the rows within the boundaries, extracts the percentage value from the specific column(column 14) as a string, removes the quotation marks around it, converts it to a double, and if it's non-zero, adds it to the sum and increments the counter. Finally, it calculates the average by dividing the sum by the counter, and prints the province name and its corresponding average as a percentage.

**b)**



```
// Average values of Canada(excluding territories)
sum=0;
counter =0;
for (i = 1; i <= 42; i++) {

    //removes the quotation marks around the strings
    char temp[MAX_LINE_LENGTH];
    strncpy(temp, array[i][13] + 1, strlen(array[i][13]) - 2);
    temp[strlen(array[i][13]) - 2] = '\0';

    double value = atof(temp);


    if(value != 0){


            sum+=value;
            counter++;
        }

}
avgCA = sum/counter;
printf("\nThe national average for Canada(excluding territories): %lf %%\n\n\n" , avgCA);
```

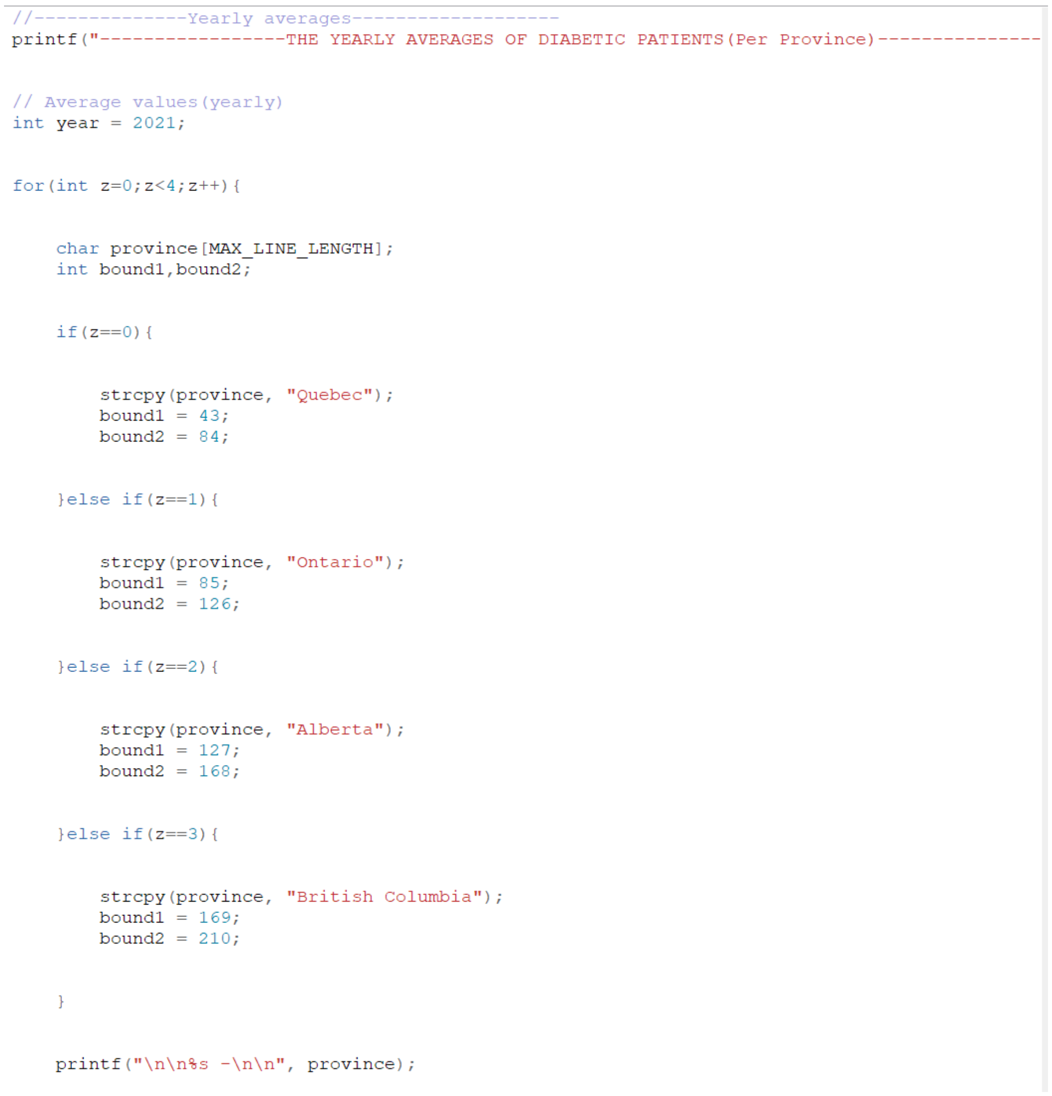
Output:



```
The national average for Canada(excluding territories): 10.869048 %
```

Similarly, the above code iterates through the specific rows for the values of Canada(excluding territories), which are 2 through 43, corresponding to 1 through 42 in the 2D array. It then extracts the percentage value from the specific column(column 14) as a string, removes the quotation marks around it, converts it to a double, and if it's non-zero, adds it to the sum and increments the counter. Finally, it calculates the average by dividing the sum by the counter, and prints the province name and its corresponding average as a percentage.

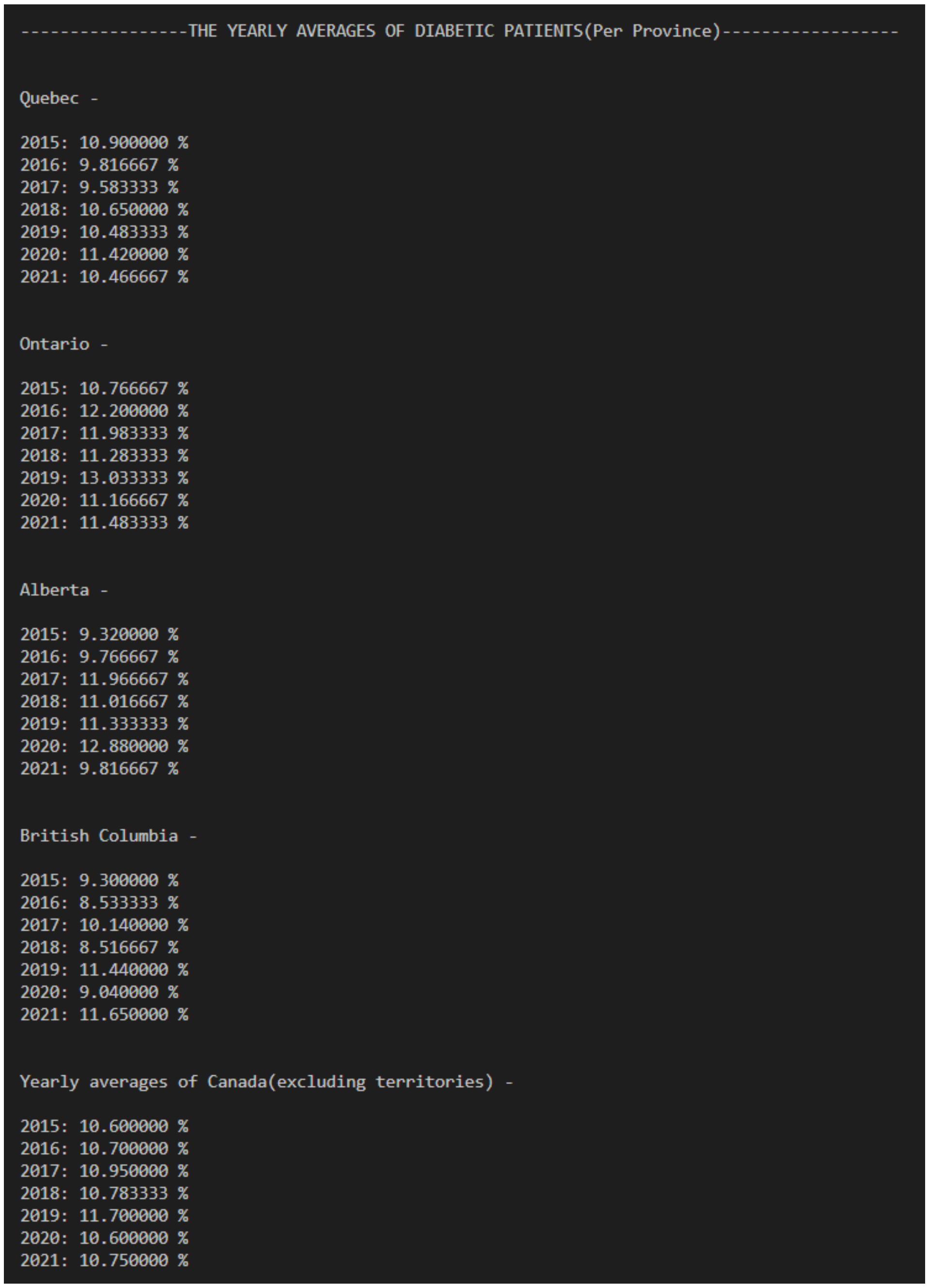
c)



```
//--------------Yearly averages-------------------
printf("-----------------THE YEARLY AVERAGES OF DIABETIC PATIENTS(Per Province)---------------


// Average values(yearly)
int year = 2021;


for(int z=0;z<4;z++){


    char province[MAX_LINE_LENGTH];
    int bound1,bound2;


    if(z==0){


        strcpy(province, "Quebec");
        bound1 = 43;
        bound2 = 84;


    }else if(z==1){


        strcpy(province, "Ontario");
        bound1 = 85;
        bound2 = 126;


    }else if(z==2){


        strcpy(province, "Alberta");
        bound1 = 127;
        bound2 = 168;


    }else if(z==3){


        strcpy(province, "British Columbia");
        bound1 = 169;
        bound2 = 210;


    }


    printf("\n\n%s -\n\n", province);
    for(int j = 2015; j<=year;j++){


        sum=0;
        counter =0;


        for (i = bound1; i <= bound2; i++) {


            //removes the quotation marks around the strings
            char temp[MAX_LINE_LENGTH];
            char temp2[MAX_LINE_LENGTH];


            strncpy(temp, array[i][13] + 1, strlen(array[i][13]) - 2);
            temp[strlen(array[i][13]) - 2] = '\0';


            strncpy(temp2, array[i][0] + 1, strlen(array[i][0]) - 2);
            temp2[strlen(array[i][0]) - 2] = '\0';


            double value = atof(temp);
            double yearval = atof(temp2);


            if(yearval == j && value != 0){

                sum+=value;
                counter++;


            }


        }


        Yavg = sum/counter;
        printf("%d: %lf %%\n" , j,Yavg);
    }
}
```

```
// Average values Canada(yearly)


printf("\n\nYearly averages of Canada(excluding territories) -\n\n");
for(int j = 2015; j<=year;j++){


    sum=0;
    counter =0;


    for (i = 1; i <= 42; i++) {


        //removes the quotation marks around the strings
        char temp[MAX_LINE_LENGTH];
        char temp2[MAX_LINE_LENGTH];


        strncpy(temp, array[i][13] + 1, strlen(array[i][13]) - 2);
        temp[strlen(array[i][13]) - 2] = '\0';


        strncpy(temp2, array[i][0] + 1, strlen(array[i][0]) - 2);
        temp2[strlen(array[i][0]) - 2] = '\0';


        double value = atof(temp);
        double yearval = atof(temp2);


        if(yearval == j && value != 0){

            sum+=value;
            counter++;


        }


    }


    YavgCA = sum/counter;
    printf("%d: %lf %%\n" , j,YavgCA);
}
```

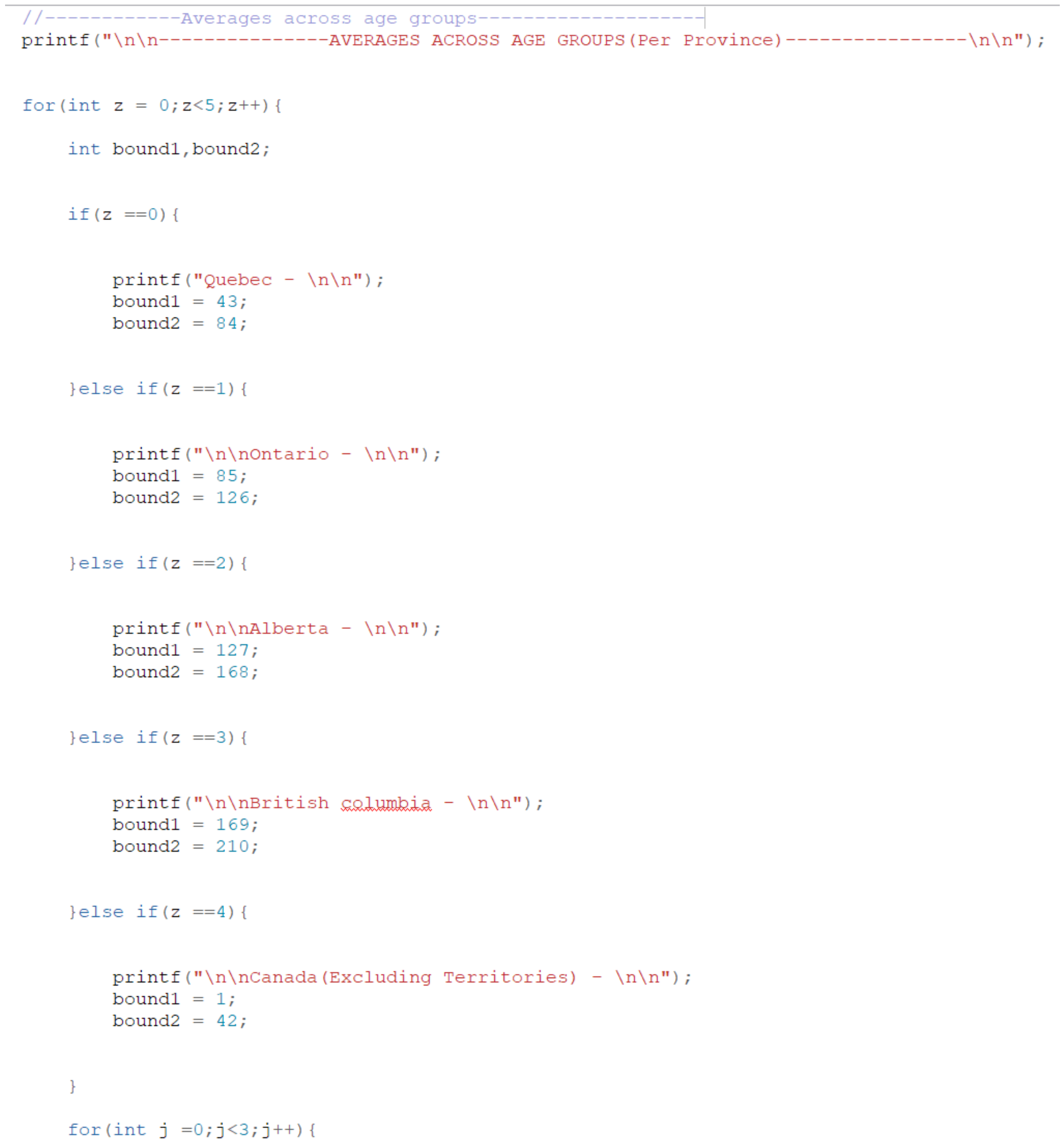
**Output:**



```
----------------THE YEARLY AVERAGES OF DIABETIC PATIENTS(Per Province)-----------------


Quebec -

2015: 10.900000 %
2016: 9.816667 %
2017: 9.583333 %
2018: 10.650000 %
2019: 10.483333 %
2020: 11.420000 %
2021: 10.466667 %


Ontario -

2015: 10.766667 %
2016: 12.200000 %
2017: 11.983333 %
2018: 11.283333 %
2019: 13.033333 %
2020: 11.166667 %
2021: 11.483333 %


Alberta -

2015: 9.320000 %
2016: 9.766667 %
2017: 11.966667 %
2018: 11.016667 %
2019: 11.333333 %
2020: 12.880000 %
2021: 9.816667 %


British Columbia -

2015: 9.300000 %
2016: 8.533333 %
2017: 10.140000 %
2018: 8.516667 %
2019: 11.440000 %
2020: 9.040000 %
2021: 11.650000 %


Yearly averages of Canada(excluding territories) -

2015: 10.600000 %
2016: 10.700000 %
2017: 10.950000 %
2018: 10.783333 %
2019: 11.700000 %
2020: 10.600000 %
2021: 10.750000 %
```

This code computes the yearly averages of diabetic patients for each province in Canada, as well as the yearly average for Canada as a whole. It first loops through each province, setting boundaries to the rows in the data array that correspond to the province. Within this loop, another loop is used to calculate the yearly average for each year from 2015 to the current year (2021 in this case). The data is filtered by year using the first column of the data array and the filtered data is used to compute the average percentage of diabetic patients for that year. The results are then printed to the console for each province and for Canada as a whole. The code uses string manipulation functions to remove quotation marks from the values read from the data file and convert them to double values.

## d)

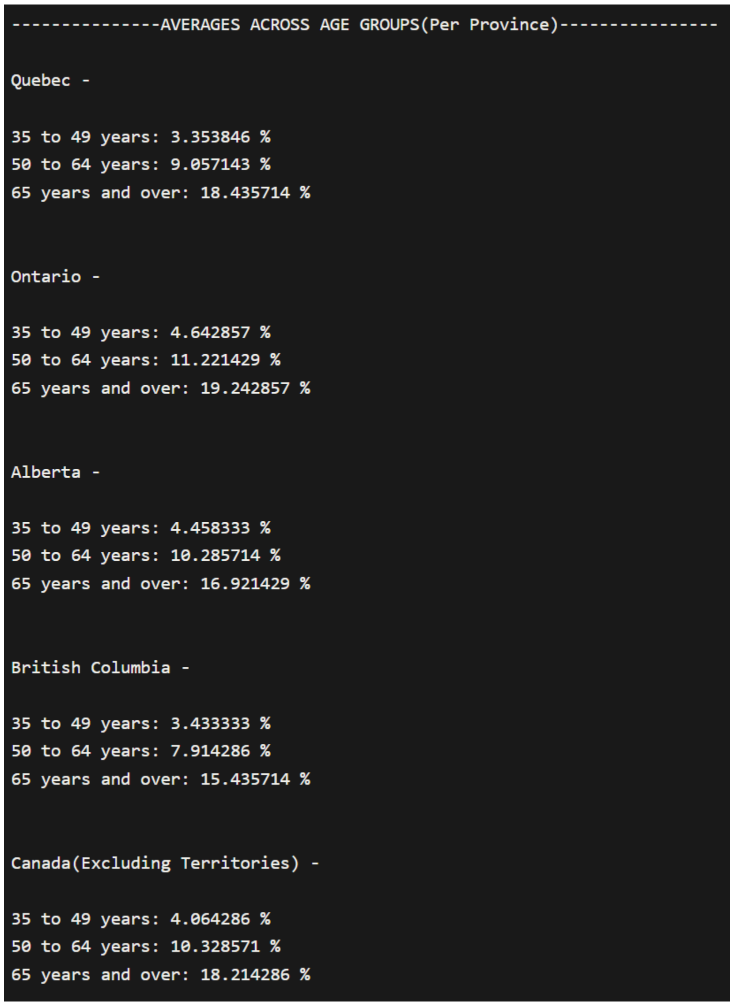


```
//------------Averages across age groups-------------------
printf("\n\n--------------AVERAGES ACROSS AGE GROUPS(Per Province)----------------\n\n");


for(int z = 0;z<5;z++){

    int bound1,bound2;


    if(z ==0){


        printf("Quebec - \n\n");
        bound1 = 43;
        bound2 = 84;


    }else if(z ==1){


        printf("\n\nOntario - \n\n");
        bound1 = 85;
        bound2 = 126;


    }else if(z ==2){


        printf("\n\nAlberta - \n\n");
        bound1 = 127;
        bound2 = 168;


    }else if(z ==3){


        printf("\n\nBritish columbia - \n\n");
        bound1 = 169;
        bound2 = 210;


    }else if(z ==4){


        printf("\n\nCanada(Excluding Territories) - \n\n");
        bound1 = 1;
        bound2 = 42;


    }

    for(int j =0;j<3;j++){
```

```
sum=0;
counter=0;
char ageGval[MAX_LINE_LENGTH];
char age1[MAX_LINE_LENGTH];
char age2[MAX_LINE_LENGTH];
char age3[MAX_LINE_LENGTH];


for (i = bound1; i <= bound2; i++) {


    //removes the quotation marks around the strings
    char temp[MAX_LINE_LENGTH];
    char temp2[MAX_LINE_LENGTH];


    strncpy(temp, array[i][13] + 1, strlen(array[i][13]) - 2);
    temp[strlen(array[i][13]) - 2] = '\0';


    strncpy(temp2, array[i][3] + 1, strlen(array[i][3]) - 2);
    temp2[strlen(array[i][3]) - 2] = '\0';


    double value = atof(temp);
    strcpy(ageGval , temp2);


    if(j == 0 && strcmp(ageGval , "35 to 49 years") == 0 && value != 0){

        sum+=value;
        counter++;
        strcpy(age1 , ageGval);


    }else if(j == 1 && strcmp(ageGval , "50 to 64 years") == 0 && value != 0){


        sum+=value;
        counter++;
        strcpy(age2 , ageGval);


    }else if(j == 2 && strcmp(ageGval , "65 years and over") == 0 && value != 0){


        sum+=value;
        counter++;
        strcpy(age3 , ageGval);


    }
```

```
            }
            Aavg = sum/counter;
            if(j == 0){


                printf("%s: %lf %%\n" , age1 , Aavg);


            }else if(j == 1){

                printf("%s: %lf %%\n" , age2 , Aavg);


            }else if(j == 2){

                printf("%s: %lf %%\n" , age3 , Aavg);


            }
        }
    }
```

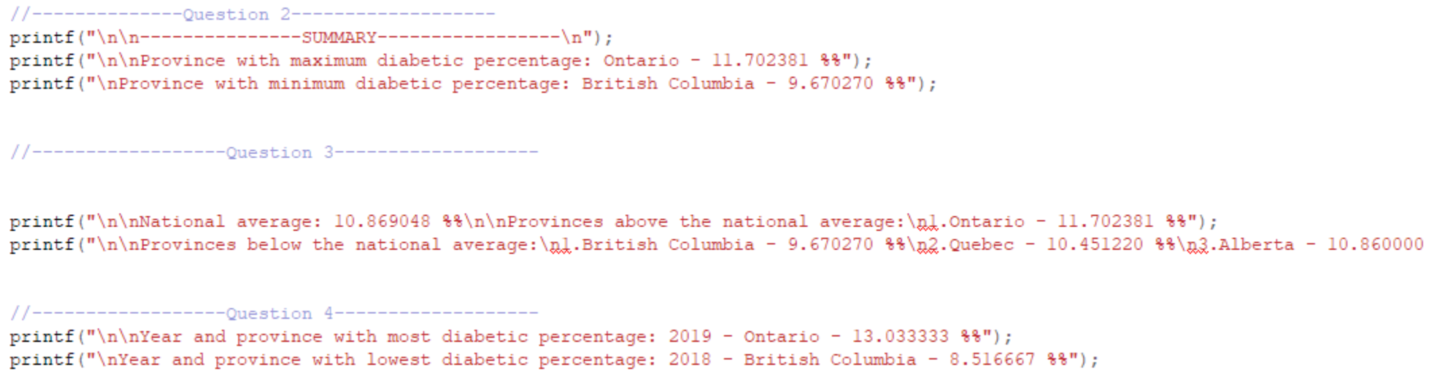
Output:



```
--------------AVERAGES ACROSS AGE GROUPS(Per Province)---------------

Quebec -

35 to 49 years: 3.353846 %
50 to 64 years: 9.057143 %
65 years and over: 18.435714 %


Ontario -

35 to 49 years: 4.642857 %
50 to 64 years: 11.221429 %
65 years and over: 19.242857 %


Alberta -

35 to 49 years: 4.458333 %
50 to 64 years: 10.285714 %
65 years and over: 16.921429 %


British Columbia -

35 to 49 years: 3.433333 %
50 to 64 years: 7.914286 %
65 years and over: 15.435714 %


Canada(Excluding Territories) -

35 to 49 years: 4.064286 %
50 to 64 years: 10.328571 %
65 years and over: 18.214286 %
```

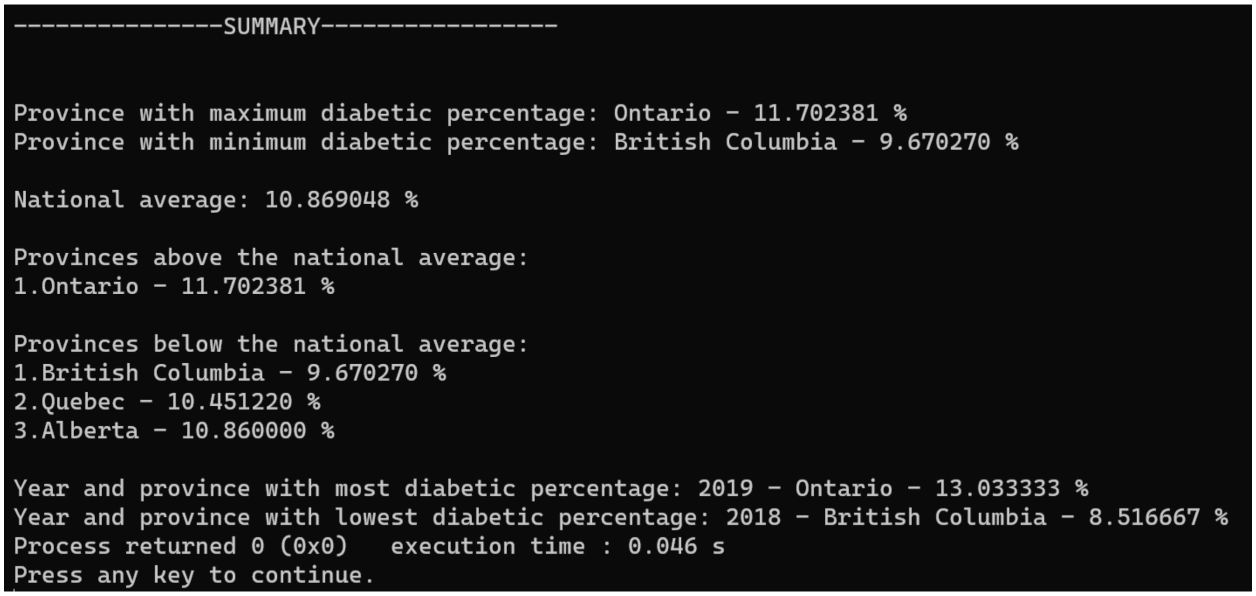
This code calculates the averages across age groups for each province in Canada using the data stored in the 2D array. It does this by first defining age group boundaries for each province, then iterating through each age group (35 to 49 years, 50 to 64 years, and 65 years and over) for each province. Within each age group iteration, the code calculates the average value for the specified age group by summing up the values for each row in the array that corresponds to that age group and province, and then dividing by the number of non-zero values found. The resulting averages are printed to the console along with the name of the age group and the corresponding province.

## Q2 & 3 & 4)



```
//--------------Question 2-------------------
printf("\n\n---------------SUMMARY-----------------\n");
printf("\n\nProvince with maximum diabetic percentage: Ontario - 11.702381 %%");
printf("\nProvince with minimum diabetic percentage: British Columbia - 9.670270 %%");


//-----------------Question 3------------------


printf("\n\nNational average: 10.869048 %%\n\nProvinces above the national average:\n1.Ontario - 11.702381 %%");
printf("\n\nProvinces below the national average:\n1.British Columbia - 9.670270 %%\n2.Quebec - 10.451220 %%\n3.Alberta - 10.860000


//-----------------Question 4------------------
printf("\n\nYear and province with most diabetic percentage: 2019 - Ontario - 13.033333 %%");
printf("\nYear and province with lowest diabetic percentage: 2018 - British Columbia - 8.516667 %%");
```

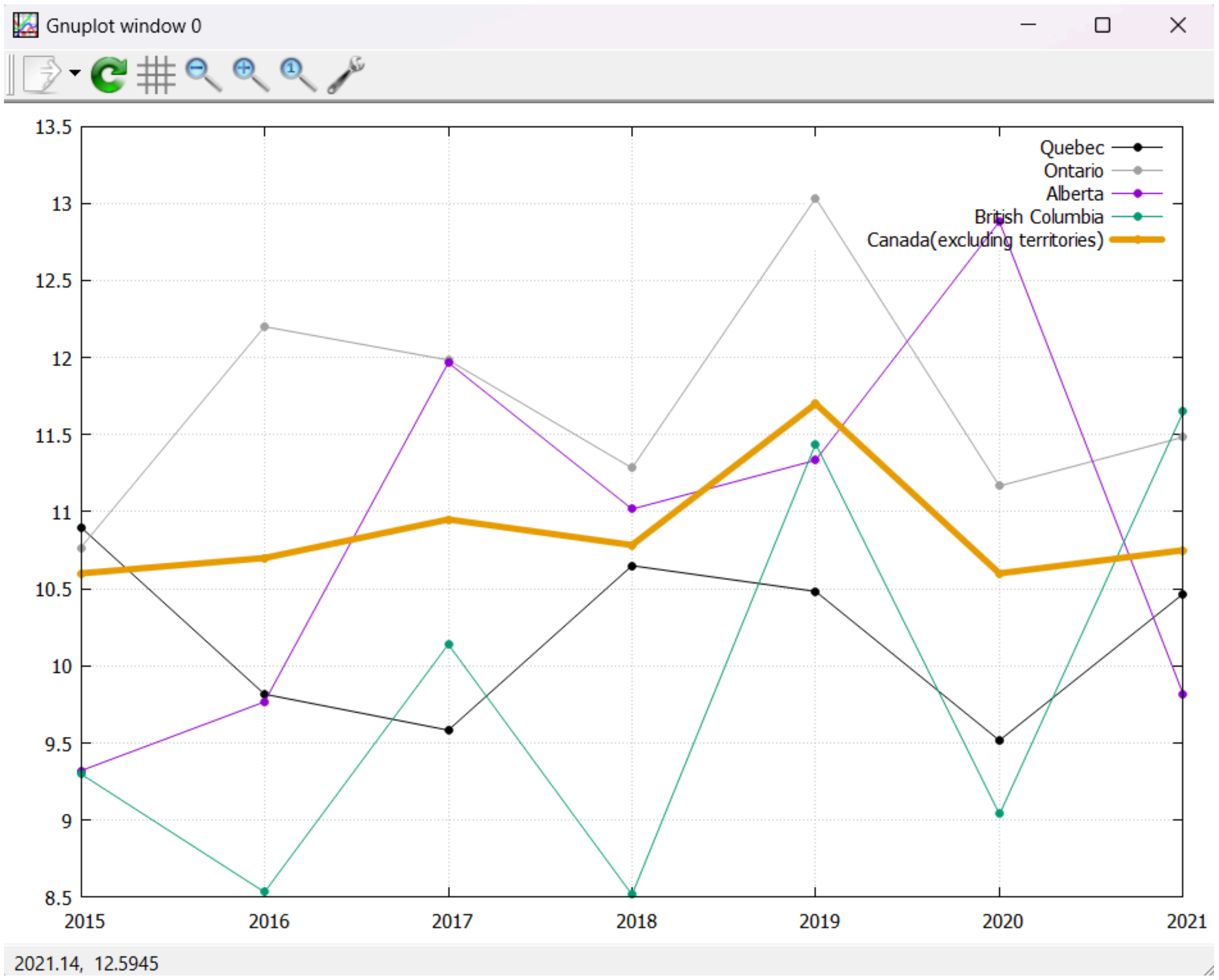
Output:



```
---------------SUMMARY-----------------


Province with maximum diabetic percentage: Ontario - 11.702381 %
Province with minimum diabetic percentage: British Columbia - 9.670270 %

National average: 10.869048 %

Provinces above the national average:
1.Ontario - 11.702381 %

Provinces below the national average:
1.British Columbia - 9.670270 %
2.Quebec - 10.451220 %
3.Alberta - 10.860000 %

Year and province with most diabetic percentage: 2019 - Ontario - 13.033333 %
Year and province with lowest diabetic percentage: 2018 - British Columbia - 8.516667 %
Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

The code contains several pairs of printf statements that display various statistical summaries related to diabetes prevalence in the different Canadian provinces. The first pair of statements show the maximum and minimum diabetic percentages among the provinces. The second pair of statements display the national average and lists the provinces that are above and below that average. The third pair of statements show the year and province with the highest and lowest diabetic percentages. Overall, the code provides valuable insights into the prevalence of diabetes in different regions of Canada.

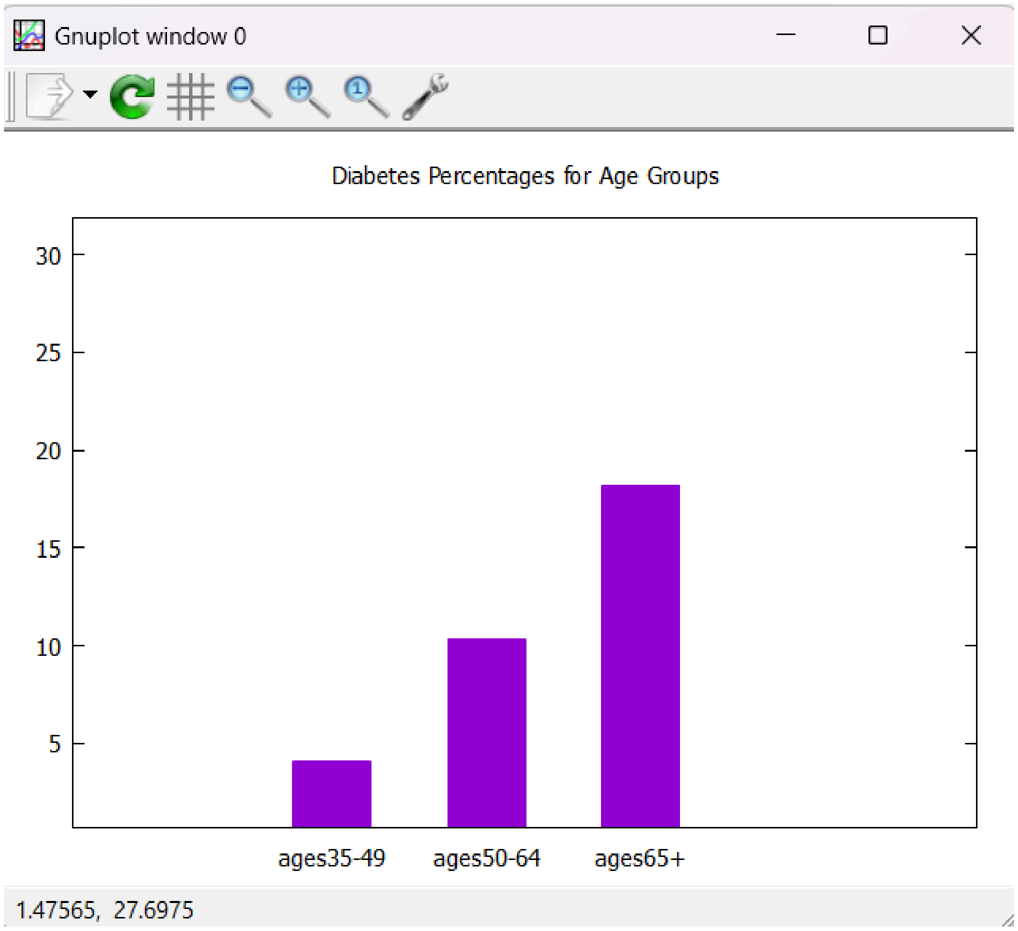
## Q5)



```
plot  "dataQ.txt" title "Quebec" lt 7 lc -1 w lp,
      "dataO.txt" title "Ontario" lt 7 lc 0 w lp,
      "dataA.txt" title "Alberta" lt 7 lc 1 w lp,
      "dataBC.txt" title "British Columbia" lt 7 lc 2 w lp,
       "dataCA.txt" title "Canada(excluding territories)" lt 7 lw 5 lc 20 w lp
```

This is a gnuplot command that plots multiple data files on a single graph. It takes five data files, each representing a province or Canada as a whole (excluding territories) and assigns a title to each. The 'lt' option is used to set the line type, and the 'lc' option is used to set the line color. The 'w' option specifies the type of line or point style to use. The 'lp' option is used to display points along the lines. The last data file, representing Canada as a whole, has a different line style and a thicker line width than the others. This command creates a line plot with multiple lines representing different provinces or territories, making it easy to compare the prevalence of diabetes across different regions.

## Q6)



```
gnuplot> set style data histograms
gnuplot> set style histogram cluster gap 1
gnuplot> set style fill solid
gnuplot> set boxwidth 0.5
gnuplot> set xtics nomirror
gnuplot> plot 'BarGraphdata.txt' using 1:3:xtic(2) with boxes
```

These gnuplot commands are used to create a clustered histogram plot from data in the file named 'BarGraphdata.txt'. The first command sets the plot style to 'histograms'. The second command sets the clustering of bars with a gap of 1. The third command sets the fill style for the bars as 'solid'. The fourth command sets the width of each bar to 0.5. The fifth command removes the mirrored x-axis tics. Finally, the plot command is used to plot the data from the file using the first column for the x-axis, the third column for the y-axis, and the second column for labeling each bar with the corresponding name, indicated by the 'xtic(2)' option. The bars are displayed using the 'boxes' style.

**Conclusion:**

With the project now concluded, we can talk about our experience working on this project.

The biggest challenge was finding a way to scan the .csv file. It took numerous attempts to scan and store the information correctly. Initially we thought that using a struct (or structs) was the way to go, but the use of a 2D array showed to be the best option. Using a struct to store and manipulate data in this case may not have been the most efficient option because structs are designed for handling a single record of data with a fixed number of fields. However, in this project, the .csv file had multiple records with varying numbers of fields.

The next hurdle we had to overcome was manipulating the data we finally had scanned and stored, because we didn't want to rewrite code for each scenario; we had to integrate the use of FOR and IF loops to streamline the code. By doing this, we were able to make the code alot tidier and efficient. For example, we assigned a number to each province, and by using a FOR loop, we went through each number (hence each province), and then used if statements to manipulate the data appropriately, and print out the result for that province with its name.

The final obstacle was to make sure that missing values were not substituted as "0" in our calculations, as mandated to us. With the use of the *counter* variable, we were able to solve this problem.

If we had to do this project again, I would sketch out a more in-depth map on what we need to do, to make it easier for us to find out what code before we actually code. It took a few iterations to get the results we wanted, but at the cost of a great deal of time. Preparing before leaping into the code could've saved us a fair amount of time, and may have resulted in a more polished code in the end. For example, use of the string.h header and its function, or the use of GNU plots. Recognizing and breaking down the tasks required for us to do in this project into smaller chunks we can focus down on would've cost less time in the end.