CPS 188 Term Project
Winter 2023

***Instructor**: Dr. Ufkes*

***TA**: Mohammed Emrul Hasan*

Sayeed Ahamad, Qurrat-Ul-Ain,
Nourhan Antar, Tre Spencer

***Student Number**: 501209136, 501169908,*
501196794, 501087581

1 Introduction

This report examines actual data gathered by Statistics Canada on the prevalence of diabetes in the four most populous provinces of Canada (Ontario, Quebec, British Columbia, and Alberta) and the as well as national data, from 2015 to 2021. The report gives a summary of a C programming project that includes computations, the creation of graphs, and conclusions based on the gathered data.

The project requires the use of the C programming language in order to take data from a CSV file, do computations, and provide the required output, including tables and graphs. The data file includes information on the prevalence of diabetes among people aged 35 and older in each of the four provinces as well as across the entire country (excluding territories).

In-depth discussion of the project's essential elements is provided in the report, including computation of annual averages, identification of the provinces with the highest and lowest percentages of diabetics, and computation of the provincial and national averages of the population with diabetes diagnoses. The report also highlights the necessity to identify the provinces above and below the national average as well as the years with the highest and lowest percentages of diabetes.

The project also requires the development of two graphs: a line plot showing diabetes percentages from 2015 to 2021 and a bar graph showing the average percentages of diabetes among the three age groups for the entire country. The study emphasizes the significance of clearly labeling the axes as well as presenting each graph with a title and a legend.

The overall objective of the project is to use C programming and GNUPlot features to investigate the prevalence of diabetes in Canada's four most populous provinces and draw conclusions using data collected by Statistics Canada.

2 Problem Sets

2.1 Problem 1

2.1.1 Computer Program

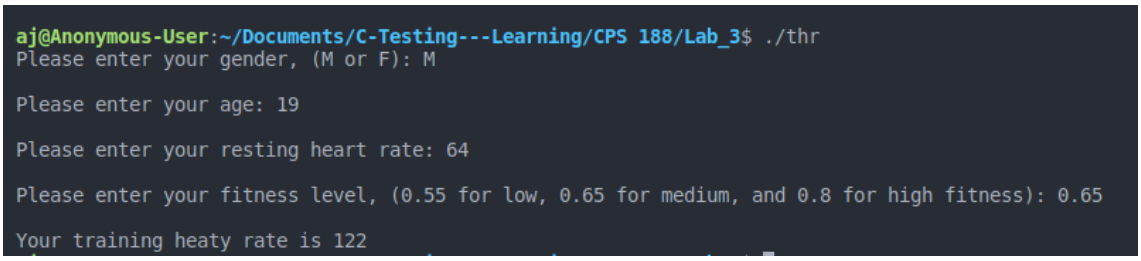
```c
/*  Program to calculate the Training Heart Rate (THR)  */

#include <stdio.h>
#include <math.h>
#include <stdbool.h>

float inputs(void);
bool gender_conditional(char gender);
int male_training_heart_rate(int age, int resting_heart_rate,
    float fitness_level);
int female_training_heart_rate(int age, int
   resting_heart_rate, float fitness_level);
int conditional(char gender, int age, int resting_heart_rate,
    float fitness_level);
void output(int training_heart_rate);

void main(void)
{
    float g, a, rhr, fl = inputs();
    int thr = conditional(g, a, rhr, fl);
    output(thr);
}

float inputs(void)
{
    char gender;
    int age;
    int resting_heart_rate;
    float fitness_level;

    /*  Scanning values for gender selection    */
    printf("Please enter your gender, (M or F): ");
    do
    {
        scanf("%c", &gender);
    } while (gender == 'M' || gender == 'F');


    /*  Scanning values for the age */
    printf("\nPlease enter your age: ");
    scanf("%i", &age);

    /*  Scanning values for the resting heart rate  */
    printf("\nPlease enter your resting heart rate: ");
    scanf("%i", &resting_heart_rate);

    /*  Scanning values for fitness level   */
    printf("\nPlease enter your fitness level, (0.55 for low,
    0.65 for medium, and 0.8 for high fitness): ");
```

```
46      scanf("%f", &fitness_level);
47
48      return gender, age, resting_heart_rate, fitness_level;
49  }
50
51  int conditional(char gender, int age, int resting_heart_rate,
        float fitness_level)
52  {
53      /*  Conditional to check male or female */
54      bool binary = gender_conditional(gender);
55
56      /*  Conditional for check male or female THR    */
57      int training_heart_rate;
58      if (binary == true)
59      {
60          training_heart_rate = male_training_heart_rate(age,
       resting_heart_rate, fitness_level);
61      }
62
63      else
64      {
65          training_heart_rate = female_training_heart_rate(age,
        resting_heart_rate, fitness_level);
66      }
67
68      return training_heart_rate;
69  }
70
71  void output(int training_heart_rate)
72  {
73      printf("\nYour training heaty rate is %i\n",
       training_heart_rate);
74  }
75
76  bool gender_conditional(char gender)
77  {
78      int binary;
79      if (gender == 'M')
80      {
81          binary = true;
82      }
83
84      else
85      {
86          binary = false;
87      }
88
89      return binary;
90  }
```

```c

int male_training_heart_rate(int age, int resting_heart_rate,
    float fitness_level)
{
    /*  Calculating the maximum heart rate  */
    float maximum_heart_rate = 203.7 / (1 + exp(0.033 * (age
   - 104.3)));

    /*  Calculating the training heart rate */
    int training_heart_rate = (maximum_heart_rate -
   resting_heart_rate) * fitness_level + resting_heart_rate;

    return training_heart_rate;
}

int female_training_heart_rate(int age, int
   resting_heart_rate, float fitness_level)
{
    /*  Calculating the maximum heart rate  */
    int maximum_heart_rate = 190.2 / (1 + exp(0.0453 * (age -
    107.5)));

    /*  Calculating the training heart rate */
    int training_heart_rate = (maximum_heart_rate -
   resting_heart_rate) * fitness_level + resting_heart_rate;

    return training_heart_rate;
}
```

Listing 2.1: *Hello World Program*

2.1.2 Program Output Screenshot



2.2 Problem 2

2.2.1 Computer Program

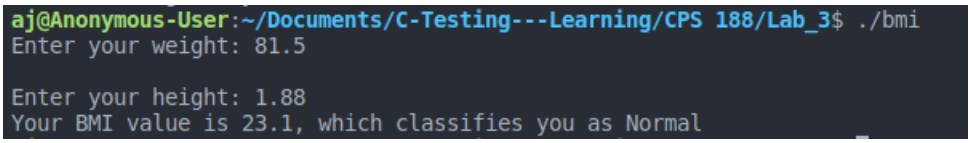
```c
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
```

```
4
5  float weight_input(void);
6  float height_input(void);
7  void output(float weight, float height);
8
9  void main(void)
10 {
11     float w = weight_input();
12     float h = height_input();
13     output(w, h);
14 }
15
16 float weight_input(void)
17 {
18     float weight;
19
20     /*  Scanning values for weight  */
21     printf("Enter your weight: ");
22     scanf("%f", &weight);
23
24     return weight;
25 }
26
27 float height_input(void)
28 {
29     float height;
30
31     /*  Scanning values for height  */
32     printf("\nEnter your height: ");
33     scanf("%f", &height);
34
35     return height;
36 }
37
38 void output(float weight, float height)
39 {
40     /*  Calculating BMI */
41     height *= height;
42     float body_mass_index = weight / (height);
43
44     /*  Conditional */
45     if (body_mass_index < 18.5)
46     {
47         printf("Your BMI value is %.1f, which classifies you
       as Underweight\n", body_mass_index);
48     }
49     else if (body_mass_index <= 24.9)
50     {
```

```
51          printf("Your BMI value is %.1f, which classifies you
    as Normal\n", body_mass_index);
52      }
53      else if (body_mass_index <= 29.9)
54      {
55          printf("Your BMI value is %.1f, which classifies you
    as Overweight\n", body_mass_index);
56      }
57      else
58      {
59          printf("Your BMI value is %.1f, which classifies you
    as Obese\n", body_mass_index);
60      }
61  }
62
```

Listing 2.2: *Program to Calculate the Body Mass Index (BMI) of a person*

2.2.2 Program Output Screenshot



2.3 Problem 3

2.3.1 Computer Program

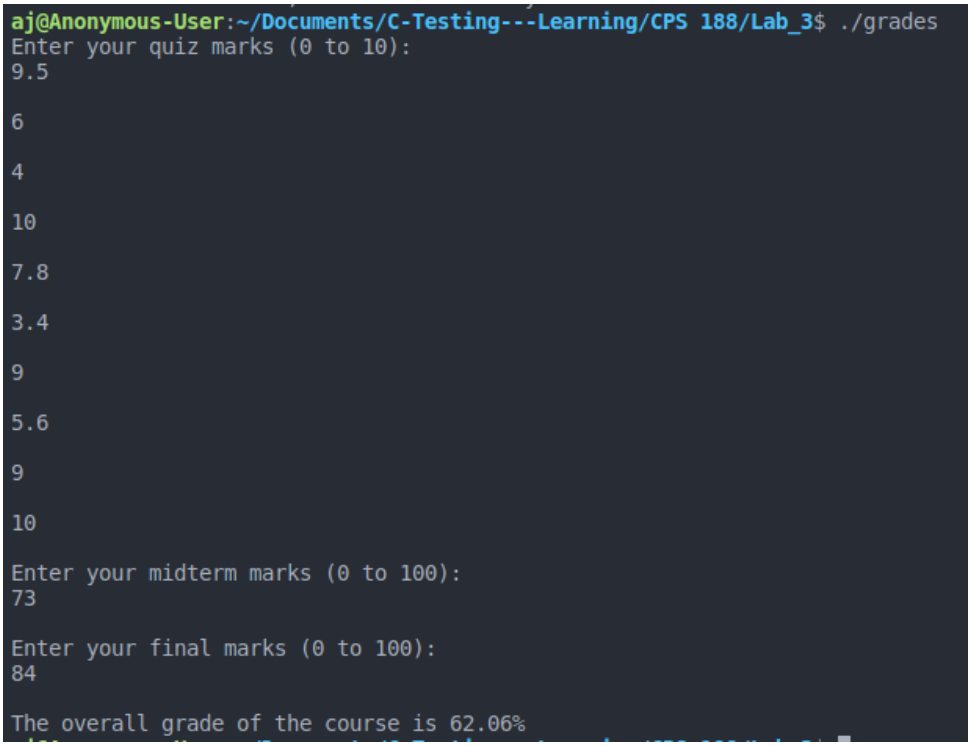
```
1   /*  Program to Calculate the Overall grades of a Course */
2
3   #include <stdio.h>
4   #include <math.h>
5
6   float quiz(void);
7   float midterm(void);
8   float final(void);
9   float conditional_output(float quiz, float midterm, float
    final);
10
11  void main(void)
12  {
13      float q = quiz();
14      float m = midterm();
15      float f = final();
16      conditional_output(q, m, f);
17  }
18
```

```c
float quiz(void)
{
    float quiz[10];
    float lowest;
    float sum = 0;

    printf("Enter your quiz marks (0 to 10):\n");
    for (int i = 0; i < 10; i++)
    {
        do
        {
            scanf("%f", &quiz[i]);
            printf("\n");
        } while (quiz[i] < 0 || quiz[i] > 10);
    }

    for (int i = 0; i < 10; i++)
    {
        if (quiz[i] < quiz[i+1])
        {
            lowest = quiz[i];
        }
    }

    for (int i = 0; i < 10; i++)
    {
        sum += quiz[i];
    }

    float average = (sum - lowest) / 9;

    return average;
}

float midterm(void)
{
    float marks;

    printf("Enter your midterm marks (0 to 100):\n");
    do
    {
        scanf("%f", &marks);
        printf("\n");
    } while (marks < 0 || marks > 100);

    return marks;
}

float final(void)
```

```
68 {
69     float marks;
70 
71     printf("Enter your final marks (0 to 100):\n");
72     do
73     {
74         scanf("%f", &marks);
75         printf("\n");
76     } while (marks < 0 || marks > 100);
77 
78     return marks;
79 }
80 
81 float conditional_output(float quiz, float midterm, float
       final)
82 {
83     quiz *= 0.25;
84 
85     if (midterm >= final)
86     {
87         midterm *= 0.35;
88         final *= 0.4;
89     }
90     else
91     {
92         midterm *= 0.25;
93         final *= 0.5;
94     }
95 
96     float grade = quiz + midterm + final;
97 
98     printf("The overall grade of the course is %.2f\n", grade
       );
99 }
100 
101 
```

Listing 2.3: *Program to Calculate the Overall grades of a Course*

2.3.2 Program Output Screenshot



Appendices

A C Source Codes

A.1 QQuestions 1, 2, 3 4 Source Code

```
1
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <math.h>
7
8  /*  Defining program macros */
9
10 #define ARRAY_SIZE 500
11 #define LINE_SIZE 250
12 #define STRING_SIZE 50
13 #define SPACERS 3
14 #define SUB_SPACERS 2
15
16 /*  Initializing UDF's in program use   */
17
18 void credits(void);
19 FILE* file_o(void);
20 void file_c(FILE* file);
21 void spacer(void);
22 void sub_spacer(void);
23 void avg_province(void);
24 void avg_year(void);
25 void year_avg(double f_2015, double f_2016, double f_2017,
      double f_2018, double f_2019, double f_2020, double f_2021
      , double q_2015, double q_2016, double q_2017, double
      q_2018, double q_2019, double q_2020, double q_2021,
      double o_2015, double o_2016, double o_2017, double o_2018
      , double o_2019, double o_2020, double o_2021, double
      a_2015, double a_2016, double a_2017, double a_2018,
      double a_2019, double a_2020, double a_2021, double b_2015
      , double b_2016, double b_2017, double b_2018, double
      b_2019, double b_2020, double b_2021);
26 void avg_age(void);
27 void age_avg(double f_35, double f_50, double f_65, double
      q_35, double q_50, double q_65, double o_35, double o_50,
      double o_65, double a_35, double a_50, double a_65, double
       b_35, double b_50, double b_65);
28
29 /*  Initializing struct datatypes for CSV data  */
```

```
30
31 typedef struct {    /*  Struct to store every Parameter in a
      line as an array of tokens with respect to their
      induvidual fields */
32   char year[10];
33   char province[35];
34   char age_group[20];
35   char sex[10];
36   char values[10];
37   char temp_str[10];
38 } datatypes;    /*  Struct DataType Variable name defined as
      "datatypes"    */
39
40 void main(void)
41 {
42   credits();
43
44     FILE* f = file_o(); /*  Initializing File Operations
     */
45
46     datatypes data_set[ARRAY_SIZE];
47     char line[LINE_SIZE];
48
49     int line_count = 0; /*  Initializing Line Counter
     Variable to count Lines in the File   */
50
51     while (!feof(f))    /*  Initializing start of CORE
     program base function    */
52     {
53         if (line_count == 0)
54         {
55             fgets(line, LINE_SIZE, f);
56             line_count++;
57             continue;   /*  Parsing the first line as line 1
     encompasses labels and headers which are of no relavance
      */
58         }
59         fgets(line, LINE_SIZE, f);
60         line_count++;   /*  Line Counter Variable Update
     */
61
62         int token_count = 0;    /*  Initializing Token
     Counter Variable to count the tokens after String
     Tokenization   */
63         char* token = strtok(line, ",");    /*  Seperating
     the line string into subsequent smaller string based on
     Comma Seperation & Tokenizing a slice of string after ","
     delimiter as a parameter   */
```

```
64          strcpy(data_set[line_count].year, token);   /*
    Assigning a string value for the Year from this base
    iteration of Var(token) [NO CONDITION REQUIRED - FIRST
    FIED ENTRY IN FILE] */
65          //token_count++;  /*  Token Counter Variable Update
     */
66
67          while(token != NULL)
68          {
69              token = strtok(NULL, ",");
70              token_count++;  /*  Token Counter Variable Update
       */
71
72              /*if (token_count == 1)
73              {
74                  strcpy(data_set[line_count].province, token);
       //  Assigning a string value for Province from this
    iteration of Var(token) if conditional satisfied
75              }*/
76              if (token_count == 3)
77              {
78                  strcpy(data_set[line_count].age_group, token)
    ;  /*  Assiging a string value of Age Goup from this
    iteration of Var(token) if conditional is satisfied   */
79              }
80              else if (token_count == 4)
81              {
82                  strcpy(data_set[line_count].sex, token);
    /*  Assigning a string value of Sex from this iteration of
     Var(token) if conditional is satisfied   */
83              }
84              else if (token_count == 13)
85              {
86                  strcpy(data_set[line_count].values, token);
    /*  Assigning a string value of Values (Raw Percentage
    Floats) from this iterationm of Var(token) if condition is
     satisfied */
87              }
88              else if (token_count == 14)
89              {
90                  strcpy(data_set[line_count].temp_str, token);
       /*  Assigning a string temp trash value of string
    literal's after Var(values) to disregard "\"" delimiter
    from being concatenated into Values string [CONDITION IS
    ALWAYS SATISFIED - TOTAL TOKEN COUNT IS 19 - temp_str HAS
    FREE malloc(20) per cycle allocation]  */
91              }
92          }
93      }   /*  End of CORE program base function   */
```

```
94
95      file_c(f);  /*  Terminating File Operations */
96
97    printf("Note: All Mathematical values and operations are
       signified and represented as follows in accordance to the
       percent operater parameter. Please refer to Project
       documentation for further information.\n");
98
99      /*          Question 1          */printf("
       |-------------------------------------------------------------------------
       Question
       1-------------------------------------------------------------------------
       n");
100
101   spacer();
102   /*    Province Wise Averages    */avg_province();
103   spacer();
104
105     /*  Initializing Sum & Iterating Counter Variables for
       Province Averages Calculation  */
106
107     //  float sum_province - Variable to store the sum of the
       Value Data Points subsequently in every iteration
108     //  province_iterator_counter - Variable to calculate the
       number of iterations performed in the for loop
109
110     /*  Federal Variables   */
111
112     float sum_federal = 0;
113     int federal_iterator_counter = 0;
114
115     /*  Quebec Variables    */
116
117     float sum_quebec = 0;
118     int quebec_iterator_counter = 0;
119
120     /*  Ontario Variables   */
121
122     float sum_ontario = 0;
123     int ontario_iterator_counter = 0;
124
125     /*  Alberta Variables   */
126
127     float sum_alberta = 0;
128     int alberta_iterator_counter = 0;
129
130     /*  British Columbia Variables  */
131
132     float sum_british_columbia = 0;
```

```
133     int british_columbia_iterator_counter = 0;
134
135     /*  Federal Average Calculator  */
136
137     for (int i = 2; i < 44; i++, federal_iterator_counter++)
138     {
139       char add_val_f[10];   /*  Initializing char variable to
        copy char pointer to char variable type   */
140       char stg_val_f[10];   /*  Initializing char variable to
        store char to char recieved from char pointer */
141       strcpy(add_val_f, data_set[i].values);    /*  Copying
      to char variable from char pointer variable */
142       strcpy(stg_val_f, &add_val_f[1]); /*  Storing the char
      variable in another char variable to manipulate string
      literals    */
143       double values_federal = atof(stg_val_f);  /*
      Converting the stored char variable to a float type
      varaible data type for mathematical computational
      maniupulation */
144       if (values_federal == 0)
145       {
146         federal_iterator_counter--; /*  Fail-Safe Mechanism
      for not counting the iterations in the iterating counter
      factor if the condition is met [CONDITION IS ONLY
      SATISFIED IF THE atof FUNCTION RETURN 0, IFF THE Var(char)
       = NULL]   */
147       }
148       //printf("%.2lf\n",values_federal);   /*  Fail-Safe in
      Testing phase to verify succinctity of the values being
      read from atof function    */
149       sum_federal += values_federal;    /*  Calculation of
      Summa function of all data points being read that are not
      NULL   */
150     } double avg_federal = (sum_federal) / (
      federal_iterator_counter); printf("Federal Average: %.3lf\
      n", avg_federal); /*  Calculation of the Average function
      from the previous Summa function and iterator counter
      function as inputs    */
151
152     /*  Quebec Average Calculator   */
153
154     for (int i = 44; i < 86; i++, quebec_iterator_counter++)
155     {
156       char add_val_q[10];   /*  Initializing char variable to
        copy char pointer to char variable type   */
157       char stg_val_q[10];   /*  Initializing char variable to
        store char to char recieved from char pointer */
158       strcpy(add_val_q, data_set[i].values);    /*  Copying
      to char variable from char pointer variable */
```

```
159     strcpy(stg_val_q, &add_val_q[1]); /*  Storing the char
    variable in another char variable to manipulate string
    literals    */
160     double values_quebec = atof(stg_val_q);  /*  Converting
     the stored char variable to a float type varaible data
    type for mathematical computational maniupulation */
161     if (values_quebec == 0)
162     {
163       quebec_iterator_counter--; /*  Fail-Safe Mechanism
    for not counting the iterations in the iterating counter
    factor if the condition is met [CONDITION IS ONLY
    SATISFIED IF THE atof FUNCTION RETURN 0, IFF THE Var(char)
     = NULL]   */
164     }
165     //printf("%.2lf\n",values_quebec);   /*  Fail-Safe in
    Testing phase to verify succinctity of the values being
    read from atof function    */
166     sum_quebec += values_quebec;    /*  Calculation of
    Summa function of all data points being read that are not
    NULL   */
167    } double avg_quebec = (sum_quebec) / (
    quebec_iterator_counter); printf("Quebec Average: %.3lf\n"
    , avg_quebec);
168
169    /*  Ontario Average Calculator  */
170
171    for (int i = 86; i < 128; i++, ontario_iterator_counter
    ++)
172    {
173     char add_val_o[10];   /*  Initializing char variable to
     copy char pointer to char variable type   */
174     char stg_val_o[10];   /*  Initializing char variable to
     store char to char recieved from char pointer */
175     strcpy(add_val_o, data_set[i].values);    /*  Copying
    to char variable from char pointer variable */
176     strcpy(stg_val_o, &add_val_o[1]); /*  Storing the char
    variable in another char variable to manipulate string
    literals    */
177     double values_ontario = atof(stg_val_o);  /*
    Converting the stored char variable to a float type
    varaible data type for mathematical computational
    maniupulation */
178     if (sum_ontario == 0)
179     {
180       ontario_iterator_counter--; /*  Fail-Safe Mechanism
    for not counting the iterations in the iterating counter
    factor if the condition is met [CONDITION IS ONLY
    SATISFIED IF THE atof FUNCTION RETURN 0, IFF THE Var(char)
     = NULL]   */
```

```
181       }
182       //printf("%.2lf\n",values_ontario);   /*  Fail-Safe in
      Testing phase to verify succinctity of the values being
      read from atof function    */
183       sum_ontario += values_ontario;    /*  Calculation of
      Summa function of all data points being read that are not
      NULL   */
184     } double avg_ontario = (sum_ontario) / (
      ontario_iterator_counter); printf("Ontario Average: %.3lf\
      n", avg_ontario);
185
186     /*  Alberta Average Calculator  */
187
188     for (int i = 128; i < 170; i++, alberta_iterator_counter
      ++)
189     {
190       char add_val_a[10];   /*  Initializing char variable to
       copy char pointer to char variable type   */
191       char stg_val_a[10];   /*  Initializing char variable to
       store char to char recieved from char pointer */
192       strcpy(add_val_a, data_set[i].values);    /*  Copying
      to char variable from char pointer variable */
193       strcpy(stg_val_a, &add_val_a[1]); /*  Storing the char
      variable in another char variable to manipulate string
      literals    */
194       double values_alberta = atof(stg_val_a);  /*
      Converting the stored char variable to a float type
      varaible data type for mathematical computational
      maniupulation */
195       if (values_alberta == 0)
196       {
197         alberta_iterator_counter--; /*  Fail-Safe Mechanism
      for not counting the iterations in the iterating counter
      factor if the condition is met [CONDITION IS ONLY
      SATISFIED IF THE atof FUNCTION RETURN 0, IFF THE Var(char)
       = NULL]   */
198       }
199       //printf("%.2lf\n",values_alberta);   /*  Fail-Safe in
      Testing phase to verify succinctity of the values being
      read from atof function    */
200       sum_alberta += values_alberta;    /*  Calculation of
      Summa function of all data points being read that are not
      NULL   */
201     } double avg_alberta = (sum_alberta) / (
      alberta_iterator_counter); printf("Alberta Average: %.3lf\
      n", avg_alberta);
202
203     /*  British Columbia Average Calculator */
204
```

```
205     for (int i = 170; i < 212; i++,
        british_columbia_iterator_counter++)
206     {
207       char add_val_b[10];   /*  Initializing char variable to
        copy char pointer to char variable type   */
208       char stg_val_b[10];   /*  Initializing char variable to
        store char to char recieved from char pointer */
209       strcpy(add_val_b, data_set[i].values);    /*  Copying
        to char variable from char pointer variable */
210       strcpy(stg_val_b, &add_val_b[1]); /*  Storing the char
        variable in another char variable to manipulate string
        literals    */
211       double values_british_columbia = atof(stg_val_b);  /*
        Converting the stored char variable to a float type
        varaible data type for mathematical computational
        maniupulation */
212       if (values_british_columbia == 0)
213       {
214         british_columbia_iterator_counter--; /*  Fail-Safe
        Mechanism for not counting the iterations in the iterating
         counter factor if the condition is met [CONDITION IS ONLY
         SATISFIED IF THE atof FUNCTION RETURN 0, IFF THE Var(char
        ) = NULL]   */
215       }
216       //printf("%.2lf\n",values_british_columbia);   /*  Fail
        -Safe in Testing phase to verify succinctity of the values
         being read from atof function    */
217       sum_british_columbia += values_british_columbia;    /*
         Calculation of Summa function of all data points being
        read that are not NULL   */
218     } double avg_british_columbia = (sum_british_columbia) /
        (british_columbia_iterator_counter); printf("British
        Columbia Average: %.3lf\n", avg_british_columbia);
219
220   spacer();
221
222     /*  Year-Wise Average Calculator*/
223
224   /*    Year Wise Averages    */avg_year();
225   spacer();
226
227   /*  Initializing Sum & Iterating Counter Variables for Year
        Averages Calculation  */
228
229     //  double sum_year - Variable to store the sum of the
        Value Data Points subsequently in every iteration
230   //  double avg_year - Variable to store the average of the
        Value Data Points subsequently after all iterations
```

```
231     //  iterator_counter_year - Variable to calculate the
        number of iterations performed in the for loop
232
233   /*  2015 Variables  */
234
235     double sum_2015 = 0;
236   double avg_2015 = 0;
237   double sum_2015_f = 0;
238   double avg_2015_f = 0;
239   double sum_2015_q = 0;
240   double avg_2015_q = 0;
241   double sum_2015_o = 0;
242   double avg_2015_o = 0;
243   double sum_2015_a = 0;
244   double avg_2015_a = 0;
245   double sum_2015_b = 0;
246   double avg_2015_b = 0;
247   int iterator_counter_2015 = 0;
248   int iterator_counter_2015_f = 0;
249   int iterator_counter_2015_q = 0;
250   int iterator_counter_2015_o = 0;
251   int iterator_counter_2015_a = 0;
252   int iterator_counter_2015_b = 0;
253
254   /*  2016 Variables  */
255
256   double sum_2016 = 0;
257   double avg_2016 = 0;
258   double sum_2016_f = 0;
259   double avg_2016_f = 0;
260   double sum_2016_q = 0;
261   double avg_2016_q = 0;
262   double sum_2016_o = 0;
263   double avg_2016_o = 0;
264   double sum_2016_a = 0;
265   double avg_2016_a = 0;
266   double sum_2016_b = 0;
267   double avg_2016_b = 0;
268   int iterator_counter_2016 = 0;
269   int iterator_counter_2016_f = 0;
270   int iterator_counter_2016_q = 0;
271   int iterator_counter_2016_o = 0;
272   int iterator_counter_2016_a = 0;
273   int iterator_counter_2016_b = 0;
274
275   /*  2017 Variables  */
276
277   double sum_2017 = 0;
278   double avg_2017 = 0;
```

```
279   double sum_2017_f = 0;
280   double avg_2017_f = 0;
281   double sum_2017_q = 0;
282   double avg_2017_q = 0;
283   double sum_2017_o = 0;
284   double avg_2017_o = 0;
285   double sum_2017_a = 0;
286   double avg_2017_a = 0;
287   double sum_2017_b = 0;
288   double avg_2017_b = 0;
289   int iterator_counter_2017 = 0;
290   int iterator_counter_2017_f = 0;
291   int iterator_counter_2017_q = 0;
292   int iterator_counter_2017_o = 0;
293   int iterator_counter_2017_a = 0;
294   int iterator_counter_2017_b = 0;
295
296   /*  2018 Variables  */
297
298   double sum_2018 = 0;
299   double avg_2018 = 0;
300   double sum_2018_f = 0;
301   double avg_2018_f = 0;
302   double sum_2018_q = 0;
303   double avg_2018_q = 0;
304   double sum_2018_o = 0;
305   double avg_2018_o = 0;
306   double sum_2018_a = 0;
307   double avg_2018_a = 0;
308   double sum_2018_b = 0;
309   double avg_2018_b = 0;
310   int iterator_counter_2018 = 0;
311   int iterator_counter_2018_f = 0;
312   int iterator_counter_2018_q = 0;
313   int iterator_counter_2018_o = 0;
314   int iterator_counter_2018_a = 0;
315   int iterator_counter_2018_b = 0;
316
317   /*  2019 Variables  */
318
319   double sum_2019 = 0;
320   double avg_2019 = 0;
321   double sum_2019_f = 0;
322   double avg_2019_f = 0;
323   double sum_2019_q = 0;
324   double avg_2019_q = 0;
325   double sum_2019_o = 0;
326   double avg_2019_o = 0;
327   double sum_2019_a = 0;
```

```
328   double avg_2019_a = 0;
329   double sum_2019_b = 0;
330   double avg_2019_b = 0;
331   int iterator_counter_2019 = 0;
332   int iterator_counter_2019_f = 0;
333   int iterator_counter_2019_q = 0;
334   int iterator_counter_2019_o = 0;
335   int iterator_counter_2019_a = 0;
336   int iterator_counter_2019_b = 0;
337
338   /*  2020 Variables  */
339
340   double sum_2020 = 0;
341   double avg_2020 = 0;
342   double sum_2020_f = 0;
343   double avg_2020_f = 0;
344   double sum_2020_q = 0;
345   double avg_2020_q = 0;
346   double sum_2020_o = 0;
347   double avg_2020_o = 0;
348   double sum_2020_a = 0;
349   double avg_2020_a = 0;
350   double sum_2020_b = 0;
351   double avg_2020_b = 0;
352   int iterator_counter_2020 = 0;
353   int iterator_counter_2020_f = 0;
354   int iterator_counter_2020_q = 0;
355   int iterator_counter_2020_o = 0;
356   int iterator_counter_2020_a = 0;
357   int iterator_counter_2020_b = 0;
358
359   /*  2021 Variables  */
360
361   double sum_2021 = 0;
362   double avg_2021 = 0;
363   double sum_2021_f = 0;
364   double avg_2021_f = 0;
365   double sum_2021_q = 0;
366   double avg_2021_q = 0;
367   double sum_2021_o = 0;
368   double avg_2021_o = 0;
369   double sum_2021_a = 0;
370   double avg_2021_a = 0;
371   double sum_2021_b = 0;
372   double avg_2021_b = 0;
373   int iterator_counter_2021 = 0;
374   int iterator_counter_2021_f = 0;
375   int iterator_counter_2021_q = 0;
376   int iterator_counter_2021_o = 0;
```

```
377   int iterator_counter_2021_a = 0;
378   int iterator_counter_2021_b = 0;
379
380     for (int i = 0; i < ARRAY_SIZE; i++)
381     {
382     for (int j = 2; j < 44; j++)  /*  Federal Year-Wise
       Average Calculator  */
383     {
384       char add_val_y[10];
385       char stg_val_y[10];
386       strcpy(add_val_y, data_set[j].year);
387       strcpy(stg_val_y, &add_val_y[1]);
388       double year = atof(stg_val_y);
389
390       if (year == 2015)
391       {
392         iterator_counter_2015_f++;
393         char add_val_2015[10];
394         char stg_val_2015[10];
395         strcpy(add_val_2015, data_set[j].values);
396         strcpy(stg_val_2015, &add_val_2015[1]);
397         double values_2015 = atof(stg_val_2015);
398         if (values_2015 == 0)
399         {
400           iterator_counter_2015_f--;
401         }
402
403         sum_2015_f += values_2015;
404       } avg_2015_f = sum_2015_f / iterator_counter_2015_f;
405       if (year == 2016)
406       {
407         iterator_counter_2016_f++;
408         char add_val_2016[10];
409         char stg_val_2016[10];
410         strcpy(add_val_2016, data_set[j].values);
411         strcpy(stg_val_2016, &add_val_2016[1]);
412         double values_2016 = atof(stg_val_2016);
413         if (values_2016 == 0)
414         {
415           iterator_counter_2016_f--;
416         }
417
418         sum_2016_f += values_2016;
419       } avg_2016_f = sum_2016_f / iterator_counter_2016_f;
420       if (year == 2017)
421       {
422         iterator_counter_2017_f++;
423         char add_val_2017[10];
424         char stg_val_2017[10];
```

```
425         strcpy(add_val_2017, data_set[j].values);
426         strcpy(stg_val_2017, &add_val_2017[1]);
427         double values_2017 = atof(stg_val_2017);
428         if (values_2017 == 0)
429         {
430           iterator_counter_2017_f--;
431         }
432 
433         sum_2017_f += values_2017;
434       } avg_2017_f = sum_2017_f / iterator_counter_2017_f;
435       if (year == 2018)
436       {
437         iterator_counter_2018_f++;
438         char add_val_2018[10];
439         char stg_val_2018[10];
440         strcpy(add_val_2018, data_set[j].values);
441         strcpy(stg_val_2018, &add_val_2018[1]);
442         double values_2018 = atof(stg_val_2018);
443         if (values_2018 == 0)
444         {
445           iterator_counter_2018_f--;
446         }
447 
448         sum_2018_f += values_2018;
449       } avg_2018_f = sum_2018_f / iterator_counter_2018_f;
450       if (year == 2019)
451       {
452         iterator_counter_2019_f++;
453         char add_val_2019[10];
454         char stg_val_2019[10];
455         strcpy(add_val_2019, data_set[j].values);
456         strcpy(stg_val_2019, &add_val_2019[1]);
457         double values_2019 = atof(stg_val_2019);
458         if (values_2019 == 0)
459         {
460           iterator_counter_2019_f--;
461         }
462 
463         sum_2019_f += values_2019;
464       } avg_2019_f = sum_2019_f / iterator_counter_2019_f;
465       if (year == 2020)
466       {
467         iterator_counter_2020_f++;
468         char add_val_2020[10];
469         char stg_val_2020[10];
470         strcpy(add_val_2020, data_set[j].values);
471         strcpy(stg_val_2020, &add_val_2020[1]);
472         double values_2020 = atof(stg_val_2020);
473         if (values_2020 == 0)
```

```
474         {
475           iterator_counter_2020_f--;
476         }
477 
478         sum_2020_f += values_2020;
479       } avg_2020_f = sum_2020_f / iterator_counter_2020_f;
480       if (year == 2021)
481       {
482         iterator_counter_2021_f++;
483         char add_val_2021[10];
484         char stg_val_2021[10];
485         strcpy(add_val_2021, data_set[j].values);
486         strcpy(stg_val_2021, &add_val_2021[1]);
487         double values_2021 = atof(stg_val_2021);
488         if (values_2021 == 0)
489         {
490           iterator_counter_2021_f--;
491         }
492 
493         sum_2021_f += values_2021;
494       } avg_2021_f = sum_2021_f / iterator_counter_2021_f;
495     }
496     for (int j = 44; j < 86; j++) /*  Quebec Year-Wise
    Average Calculator */
497     {
498       char add_val_y[10];
499       char stg_val_y[10];
500       strcpy(add_val_y, data_set[j].year);
501       strcpy(stg_val_y, &add_val_y[1]);
502       double year = atof(stg_val_y);
503 
504       if (year == 2015)
505       {
506         iterator_counter_2015_q++;
507         char add_val_2015[10];
508         char stg_val_2015[10];
509         strcpy(add_val_2015, data_set[j].values);
510         strcpy(stg_val_2015, &add_val_2015[1]);
511         double values_2015 = atof(stg_val_2015);
512         if (values_2015 == 0)
513         {
514           iterator_counter_2015_q--;
515         }
516 
517         sum_2015_q += values_2015;
518       } avg_2015_q = sum_2015_q / iterator_counter_2015_q;
519       if (year == 2016)
520       {
521         iterator_counter_2016_q++;
```

```
522         char add_val_2016[10];
523         char stg_val_2016[10];
524         strcpy(add_val_2016, data_set[j].values);
525         strcpy(stg_val_2016, &add_val_2016[1]);
526         double values_2016 = atof(stg_val_2016);
527         if (values_2016 == 0)
528         {
529           iterator_counter_2016_q--;
530         }
531 
532         sum_2016_q += values_2016;
533       } avg_2016_q = sum_2016_q / iterator_counter_2016_q;
534       if (year == 2017)
535       {
536         iterator_counter_2017_q++;
537         char add_val_2017[10];
538         char stg_val_2017[10];
539         strcpy(add_val_2017, data_set[j].values);
540         strcpy(stg_val_2017, &add_val_2017[1]);
541         double values_2017 = atof(stg_val_2017);
542         if (values_2017 == 0)
543         {
544           iterator_counter_2017_q--;
545         }
546 
547         sum_2017_q += values_2017;
548       } avg_2017_q = sum_2017_q / iterator_counter_2017_q;
549       if (year == 2018)
550       {
551         iterator_counter_2018_q++;
552         char add_val_2018[10];
553         char stg_val_2018[10];
554         strcpy(add_val_2018, data_set[j].values);
555         strcpy(stg_val_2018, &add_val_2018[1]);
556         double values_2018 = atof(stg_val_2018);
557         if (values_2018 == 0)
558         {
559           iterator_counter_2018_q--;
560         }
561 
562         sum_2018_q += values_2018;
563       } avg_2018_q = sum_2018_q / iterator_counter_2018_q;
564       if (year == 2019)
565       {
566         iterator_counter_2019_q++;
567         char add_val_2019[10];
568         char stg_val_2019[10];
569         strcpy(add_val_2019, data_set[j].values);
570         strcpy(stg_val_2019, &add_val_2019[1]);
```

```
571         double values_2019 = atof(stg_val_2019);
572         if (values_2019 == 0)
573         {
574           iterator_counter_2019_q--;
575         }
576 
577         sum_2019 += values_2019;
578       } avg_2019_q = sum_2019_q / iterator_counter_2019_q;
579       if (year == 2020)
580       {
581         iterator_counter_2020_q++;
582         char add_val_2020[10];
583         char stg_val_2020[10];
584         strcpy(add_val_2020, data_set[j].values);
585         strcpy(stg_val_2020, &add_val_2020[1]);
586         double values_2020 = atof(stg_val_2020);
587         if (values_2020 == 0)
588         {
589           iterator_counter_2020_q--;
590         }
591 
592         sum_2020_q += values_2020;
593       } avg_2020_q = sum_2020_q / iterator_counter_2020_q;
594       if (year == 2021)
595       {
596         iterator_counter_2021_q++;
597         char add_val_2021[10];
598         char stg_val_2021[10];
599         strcpy(add_val_2021, data_set[j].values);
600         strcpy(stg_val_2021, &add_val_2021[1]);
601         double values_2021 = atof(stg_val_2021);
602         if (values_2021 == 0)
603         {
604           iterator_counter_2021_q--;
605         }
606 
607         sum_2021_q += values_2021;
608       } avg_2021_q = sum_2021_q / iterator_counter_2021_q;
609     }
610     for (int j = 86; j < 128; j++)  /*  Ontario Year-Wise
    Average Calculator  */
611     {
612       char add_val_y[10];
613       char stg_val_y[10];
614       strcpy(add_val_y, data_set[j].year);
615       strcpy(stg_val_y, &add_val_y[1]);
616       double year = atof(stg_val_y);
617 
618       if (year == 2015)
```

```
619       {
620         iterator_counter_2015_o++;
621         char add_val_2015[10];
622         char stg_val_2015[10];
623         strcpy(add_val_2015, data_set[j].values);
624         strcpy(stg_val_2015, &add_val_2015[1]);
625         double values_2015 = atof(stg_val_2015);
626         if (values_2015 == 0)
627         {
628           iterator_counter_2015_o--;
629         }
630 
631         sum_2015_o += values_2015;
632       } avg_2015_o = sum_2015_o / iterator_counter_2015_o;
633       if (year == 2016)
634       {
635         iterator_counter_2016_o++;
636         char add_val_2016[10];
637         char stg_val_2016[10];
638         strcpy(add_val_2016, data_set[j].values);
639         strcpy(stg_val_2016, &add_val_2016[1]);
640         double values_2016 = atof(stg_val_2016);
641         if (values_2016 == 0)
642         {
643           iterator_counter_2016_o--;
644         }
645 
646         sum_2016_o += values_2016;
647       } avg_2016_o = sum_2016_o / iterator_counter_2016_o;
648       if (year == 2017)
649       {
650         iterator_counter_2017_o++;
651         char add_val_2017[10];
652         char stg_val_2017[10];
653         strcpy(add_val_2017, data_set[j].values);
654         strcpy(stg_val_2017, &add_val_2017[1]);
655         double values_2017 = atof(stg_val_2017);
656         if (values_2017 == 0)
657         {
658           iterator_counter_2017_o--;
659         }
660 
661         sum_2017_o += values_2017;
662       } avg_2017_o = sum_2017_o / iterator_counter_2017_o;
663       if (year == 2018)
664       {
665         iterator_counter_2018_o++;
666         char add_val_2018[10];
667         char stg_val_2018[10];
```

```
668         strcpy(add_val_2018, data_set[j].values);
669         strcpy(stg_val_2018, &add_val_2018[1]);
670         double values_2018 = atof(stg_val_2018);
671         if (values_2018 == 0)
672         {
673           iterator_counter_2018_o--;
674         }
675 
676         sum_2018_o += values_2018;
677       } avg_2018_o = sum_2018_o / iterator_counter_2018_o;
678       if (year == 2019)
679       {
680         iterator_counter_2019_o++;
681         char add_val_2019[10];
682         char stg_val_2019[10];
683         strcpy(add_val_2019, data_set[j].values);
684         strcpy(stg_val_2019, &add_val_2019[1]);
685         double values_2019 = atof(stg_val_2019);
686         if (values_2019 == 0)
687         {
688           iterator_counter_2019_o--;
689         }
690 
691         sum_2019_o += values_2019;
692       } avg_2019_o = sum_2019_o / iterator_counter_2019_o;
693       if (year == 2020)
694       {
695         iterator_counter_2020_o++;
696         char add_val_2020[10];
697         char stg_val_2020[10];
698         strcpy(add_val_2020, data_set[j].values);
699         strcpy(stg_val_2020, &add_val_2020[1]);
700         double values_2020 = atof(stg_val_2020);
701         if (values_2020 == 0)
702         {
703           iterator_counter_2020_o--;
704         }
705 
706         sum_2020_o += values_2020;
707       } avg_2020_o = sum_2020_o / iterator_counter_2020_o;
708       if (year == 2021)
709       {
710         iterator_counter_2021_o++;
711         char add_val_2021[10];
712         char stg_val_2021[10];
713         strcpy(add_val_2021, data_set[j].values);
714         strcpy(stg_val_2021, &add_val_2021[1]);
715         double values_2021 = atof(stg_val_2021);
716         if (values_2021 == 0)
```

```
717         {
718           iterator_counter_2021_o--;
719         }
720 
721         sum_2021_o += values_2021;
722       } avg_2021_o = sum_2021_o / iterator_counter_2021_o;
723     }
724     for (int j = 128; j < 170; j++) /*  Alberta Year-Wise
    Average Calculator  */
725     {
726       char add_val_y[10];
727       char stg_val_y[10];
728       strcpy(add_val_y, data_set[j].year);
729       strcpy(stg_val_y, &add_val_y[1]);
730       double year = atof(stg_val_y);
731 
732       if (year == 2015)
733       {
734         iterator_counter_2015_a++;
735         char add_val_2015[10];
736         char stg_val_2015[10];
737         strcpy(add_val_2015, data_set[j].values);
738         strcpy(stg_val_2015, &add_val_2015[1]);
739         double values_2015 = atof(stg_val_2015);
740         if (values_2015 == 0)
741         {
742           iterator_counter_2015_a--;
743         }
744 
745         sum_2015_a += values_2015;
746       } avg_2015_a = sum_2015_a / iterator_counter_2015_a;
747       if (year == 2016)
748       {
749         iterator_counter_2016_a++;
750         char add_val_2016[10];
751         char stg_val_2016[10];
752         strcpy(add_val_2016, data_set[j].values);
753         strcpy(stg_val_2016, &add_val_2016[1]);
754         double values_2016 = atof(stg_val_2016);
755         if (values_2016 == 0)
756         {
757           iterator_counter_2016_a--;
758         }
759 
760         sum_2016_a += values_2016;
761       } avg_2016_a = sum_2016_a / iterator_counter_2016_a;
762       if (year == 2017)
763       {
764         iterator_counter_2017_a++;
```

```
765         char add_val_2017[10];
766         char stg_val_2017[10];
767         strcpy(add_val_2017, data_set[j].values);
768         strcpy(stg_val_2017, &add_val_2017[1]);
769         double values_2017 = atof(stg_val_2017);
770         if (values_2017 == 0)
771         {
772           iterator_counter_2017_a--;
773         }
774 
775         sum_2017_a += values_2017;
776       } avg_2017_a = sum_2017_a / iterator_counter_2017_a;
777       if (year == 2018)
778       {
779         iterator_counter_2018_a++;
780         char add_val_2018[10];
781         char stg_val_2018[10];
782         strcpy(add_val_2018, data_set[j].values);
783         strcpy(stg_val_2018, &add_val_2018[1]);
784         double values_2018 = atof(stg_val_2018);
785         if (values_2018 == 0)
786         {
787           iterator_counter_2018_a--;
788         }
789 
790         sum_2018_a += values_2018;
791       } avg_2018_a = sum_2018_a / iterator_counter_2018_a;
792       if (year == 2019)
793       {
794         iterator_counter_2019_a++;
795         char add_val_2019[10];
796         char stg_val_2019[10];
797         strcpy(add_val_2019, data_set[j].values);
798         strcpy(stg_val_2019, &add_val_2019[1]);
799         double values_2019 = atof(stg_val_2019);
800         if (values_2019 == 0)
801         {
802           iterator_counter_2019_a--;
803         }
804 
805         sum_2019_a += values_2019;
806       } avg_2019_a = sum_2019_a / iterator_counter_2019_a;
807       if (year == 2020)
808       {
809         iterator_counter_2020_a++;
810         char add_val_2020[10];
811         char stg_val_2020[10];
812         strcpy(add_val_2020, data_set[j].values);
813         strcpy(stg_val_2020, &add_val_2020[1]);
```

```
814         double values_2020 = atof(stg_val_2020);
815         if (values_2020 == 0)
816         {
817           iterator_counter_2020_a--;
818         }
819 
820         sum_2020_a += values_2020;
821       } avg_2020_a = sum_2020_a / iterator_counter_2020_a;
822       if (year == 2021)
823       {
824         iterator_counter_2021_a++;
825         char add_val_2021[10];
826         char stg_val_2021[10];
827         strcpy(add_val_2021, data_set[j].values);
828         strcpy(stg_val_2021, &add_val_2021[1]);
829         double values_2021 = atof(stg_val_2021);
830         if (values_2021 == 0)
831         {
832           iterator_counter_2021_a--;
833         }
834 
835         sum_2021_a += values_2021;
836       } avg_2021_a = sum_2021_a / iterator_counter_2021_a;
837     }
838     for (int j = 170; j < 212; j++) /*  British Columbia Year
    -Wise Average Calculator */
839     {
840       char add_val_y[10];
841       char stg_val_y[10];
842       strcpy(add_val_y, data_set[j].year);
843       strcpy(stg_val_y, &add_val_y[1]);
844       double year = atof(stg_val_y);
845 
846       if (year == 2015)
847       {
848         iterator_counter_2015_b++;
849         char add_val_2015[10];
850         char stg_val_2015[10];
851         strcpy(add_val_2015, data_set[j].values);
852         strcpy(stg_val_2015, &add_val_2015[1]);
853         double values_2015 = atof(stg_val_2015);
854         if (values_2015 == 0)
855         {
856           iterator_counter_2015_b--;
857         }
858 
859         sum_2015_b += values_2015;
860       } avg_2015_b = sum_2015_b / iterator_counter_2015_b;
861       if (year == 2016)
```

```
862         {
863           iterator_counter_2016_b++;
864           char add_val_2016[10];
865           char stg_val_2016[10];
866           strcpy(add_val_2016, data_set[j].values);
867           strcpy(stg_val_2016, &add_val_2016[1]);
868           double values_2016 = atof(stg_val_2016);
869           if (values_2016 == 0)
870           {
871             iterator_counter_2016_b--;
872           }
873
874           sum_2016_b += values_2016;
875         } avg_2016_b = sum_2016_b / iterator_counter_2016_b;
876         if (year == 2017)
877         {
878           iterator_counter_2017_b++;
879           char add_val_2017[10];
880           char stg_val_2017[10];
881           strcpy(add_val_2017, data_set[j].values);
882           strcpy(stg_val_2017, &add_val_2017[1]);
883           double values_2017 = atof(stg_val_2017);
884           if (values_2017 == 0)
885           {
886             iterator_counter_2017_b--;
887           }
888
889           sum_2017_b += values_2017;
890         } avg_2017_b = sum_2017_b / iterator_counter_2017_b;
891         if (year == 2018)
892         {
893           iterator_counter_2018_b++;
894           char add_val_2018[10];
895           char stg_val_2018[10];
896           strcpy(add_val_2018, data_set[j].values);
897           strcpy(stg_val_2018, &add_val_2018[1]);
898           double values_2018 = atof(stg_val_2018);
899           if (values_2018 == 0)
900           {
901             iterator_counter_2018_b--;
902           }
903
904           sum_2018_b += values_2018;
905         } avg_2018_b = sum_2018_b / iterator_counter_2018_b;
906         if (year == 2019)
907         {
908           iterator_counter_2019_b++;
909           char add_val_2019[10];
910           char stg_val_2019[10];
```

```
911         strcpy(add_val_2019, data_set[j].values);
912         strcpy(stg_val_2019, &add_val_2019[1]);
913         double values_2019 = atof(stg_val_2019);
914         if (values_2019 == 0)
915         {
916           iterator_counter_2019_b--;
917         }
918 
919         sum_2019_b += values_2019;
920       } avg_2019_b = sum_2019_b / iterator_counter_2019_b;
921       if (year == 2020)
922       {
923         iterator_counter_2020_b++;
924         char add_val_2020[10];
925         char stg_val_2020[10];
926         strcpy(add_val_2020, data_set[j].values);
927         strcpy(stg_val_2020, &add_val_2020[1]);
928         double values_2020 = atof(stg_val_2020);
929         if (values_2020 == 0)
930         {
931           iterator_counter_2020_b--;
932         }
933 
934         sum_2020_b += values_2020;
935       } avg_2020_b = sum_2020_b / iterator_counter_2020_b;
936       if (year == 2021)
937       {
938         iterator_counter_2021_b++;
939         char add_val_2021[10];
940         char stg_val_2021[10];
941         strcpy(add_val_2021, data_set[j].values);
942         strcpy(stg_val_2021, &add_val_2021[1]);
943         double values_2021 = atof(stg_val_2021);
944         if (values_2021 == 0)
945         {
946           iterator_counter_2021_b--;
947         }
948 
949         sum_2021_b += values_2021;
950       } avg_2021_b = sum_2021_b / iterator_counter_2021_b;
951     }
952 
953 
954     } year_avg(avg_2015_f, avg_2016_f, avg_2017_f, avg_2018_f
    , avg_2019_f, avg_2020_f, avg_2021_f, avg_2015_q,
    avg_2016_q, avg_2017_q, avg_2018_q, avg_2019_q, avg_2020_q
    , avg_2021_q, avg_2015_o, avg_2016_o, avg_2017_o,
    avg_2018_o, avg_2019_o, avg_2020_o, avg_2021_o, avg_2015_a
    , avg_2016_a, avg_2017_a, avg_2018_a, avg_2019_a,
```

```
 avg_2020_a, avg_2021_a, avg_2015_b, avg_2016_b, avg_2017_b
 , avg_2018_b, avg_2019_b, avg_2020_b, avg_2021_b);

spacer();

  /*  Age-Group-Wise Average Calculator*/

/*    Age-Group Wise Averages   */avg_age();
spacer();

/*  Initializing Sum & Iterating Counter Variables for Age-
 Group Averages Calculation */

  //  double sum_year - Variable to store the sum of the
 Value Data Points subsequently in every iteration
//  double avg_year - Variable to store the average of the
 Value Data Points subsequently after all iterations
  //  iterator_counter_year - Variable to calculate the
 number of iterations performed in the for loop

/*  35-49 Age Group Variables */

  double sum_35 = 0;
double avg_35 = 0;
double sum_35_f = 0;
double avg_35_f = 0;
double sum_35_q = 0;
double avg_35_q = 0;
double sum_35_o = 0;
double avg_35_o = 0;
double sum_35_a = 0;
double avg_35_a = 0;
double sum_35_b = 0;
double avg_35_b = 0;
int iterator_counter_35 = 0;
int iterator_counter_35_f = 0;
int iterator_counter_35_q = 0;
int iterator_counter_35_o = 0;
int iterator_counter_35_a = 0;
int iterator_counter_35_b = 0;

/*  50-65 Age Group Variables */

double sum_50 = 0;
double avg_50 = 0;
double sum_50_f = 0;
double avg_50_f = 0;
double sum_50_q = 0;
double avg_50_q = 0;
```

```
998   double sum_50_o = 0;
999   double avg_50_o = 0;
1000  double sum_50_a = 0;
1001  double avg_50_a = 0;
1002  double sum_50_b = 0;
1003  double avg_50_b = 0;
1004  int iterator_counter_50 = 0;
1005  int iterator_counter_50_f = 0;
1006  int iterator_counter_50_q = 0;
1007  int iterator_counter_50_o = 0;
1008  int iterator_counter_50_a = 0;
1009  int iterator_counter_50_b = 0;
1010
1011  /*  65+ Age Group Variables */
1012
1013  double sum_65 = 0;
1014  double avg_65 = 0;
1015  double sum_65_f = 0;
1016  double avg_65_f = 0;
1017  double sum_65_q = 0;
1018  double avg_65_q = 0;
1019  double sum_65_o = 0;
1020  double avg_65_o = 0;
1021  double sum_65_a = 0;
1022  double avg_65_a = 0;
1023  double sum_65_b = 0;
1024  double avg_65_b = 0;
1025  int iterator_counter_65 = 0;
1026  int iterator_counter_65_f = 0;
1027  int iterator_counter_65_q = 0;
1028  int iterator_counter_65_o = 0;
1029  int iterator_counter_65_a = 0;
1030  int iterator_counter_65_b = 0;
1031
1032    for (int i = 0; i < ARRAY_SIZE; i++)
1033    {
1034    for (int j = 2; j < 44; j++)  /*  Federal Age-Group-Wise
      Average Calculator */
1035    {
1036      char add_val_y[10];
1037      char stg_val_y[10];
1038      strcpy(add_val_y, data_set[j].age_group);
1039      strcpy(stg_val_y, &add_val_y[1]);
1040      double age_group = atof(stg_val_y);
1041
1042      if (age_group == 35)
1043      {
1044        iterator_counter_35_f++;
1045        char add_val_2015[10];
```

```
1046        char stg_val_2015[10];
1047        strcpy(add_val_2015, data_set[j].values);
1048        strcpy(stg_val_2015, &add_val_2015[1]);
1049        double values_2015 = atof(stg_val_2015);
1050        if (values_2015 == 0)
1051        {
1052          iterator_counter_35_f--;
1053        }
1054
1055        sum_35_f += values_2015;
1056      } avg_35_f = sum_35_f / iterator_counter_35_f;
1057      if (age_group == 50)
1058      {
1059        iterator_counter_50_f++;
1060        char add_val_2016[10];
1061        char stg_val_2016[10];
1062        strcpy(add_val_2016, data_set[j].values);
1063        strcpy(stg_val_2016, &add_val_2016[1]);
1064        double values_2016 = atof(stg_val_2016);
1065        if (values_2016 == 0)
1066        {
1067          iterator_counter_50_f--;
1068        }
1069
1070        sum_50_f += values_2016;
1071      } avg_50_f = sum_50_f / iterator_counter_50_f;
1072      if (age_group == 65)
1073      {
1074        iterator_counter_65_f++;
1075        char add_val_2017[10];
1076        char stg_val_2017[10];
1077        strcpy(add_val_2017, data_set[j].values);
1078        strcpy(stg_val_2017, &add_val_2017[1]);
1079        double values_2017 = atof(stg_val_2017);
1080        if (values_2017 == 0)
1081        {
1082          iterator_counter_65_f--;
1083        }
1084
1085        sum_65_f += values_2017;
1086      } avg_65_f = sum_65_f / iterator_counter_65_f;
1087    }
1088    for (int j = 44; j < 86; j++) /*  Quebec Age-Group-Wise
     Average Calculator  */
1089    {
1090      char add_val_y[10];
1091      char stg_val_y[10];
1092      strcpy(add_val_y, data_set[j].age_group);
1093      strcpy(stg_val_y, &add_val_y[1]);
```

```
1094        double age_group = atof(stg_val_y);
1095
1096        if (age_group == 35)
1097        {
1098          iterator_counter_35_q++;
1099          char add_val_2015[10];
1100          char stg_val_2015[10];
1101          strcpy(add_val_2015, data_set[j].values);
1102          strcpy(stg_val_2015, &add_val_2015[1]);
1103          double values_2015 = atof(stg_val_2015);
1104          if (values_2015 == 0)
1105          {
1106            iterator_counter_35_q--;
1107          }
1108
1109          sum_35_q += values_2015;
1110        } avg_35_q = sum_35_q / iterator_counter_35_q;
1111        if (age_group == 50)
1112        {
1113          iterator_counter_50_q++;
1114          char add_val_2016[10];
1115          char stg_val_2016[10];
1116          strcpy(add_val_2016, data_set[j].values);
1117          strcpy(stg_val_2016, &add_val_2016[1]);
1118          double values_2016 = atof(stg_val_2016);
1119          if (values_2016 == 0)
1120          {
1121            iterator_counter_50_q--;
1122          }
1123
1124          sum_50_q += values_2016;
1125        } avg_50_q = sum_50_q / iterator_counter_50_q;
1126        if (age_group == 65)
1127        {
1128          iterator_counter_65_q++;
1129          char add_val_2017[10];
1130          char stg_val_2017[10];
1131          strcpy(add_val_2017, data_set[j].values);
1132          strcpy(stg_val_2017, &add_val_2017[1]);
1133          double values_2017 = atof(stg_val_2017);
1134          if (values_2017 == 0)
1135          {
1136            iterator_counter_65_q--;
1137          }
1138
1139          sum_65_q += values_2017;
1140        } avg_65_q = sum_65_q / iterator_counter_65_q;
1141      }
```

```
1142     for (int j = 86; j < 128; j++)  /*  Ontario Age-Group-
    Wise Average Calculator */
1143     {
1144       char add_val_y[10];
1145       char stg_val_y[10];
1146       strcpy(add_val_y, data_set[j].age_group);
1147       strcpy(stg_val_y, &add_val_y[1]);
1148       double age_group = atof(stg_val_y);
1149 
1150       if (age_group == 35)
1151       {
1152         iterator_counter_35_o++;
1153         char add_val_2015[10];
1154         char stg_val_2015[10];
1155         strcpy(add_val_2015, data_set[j].values);
1156         strcpy(stg_val_2015, &add_val_2015[1]);
1157         double values_2015 = atof(stg_val_2015);
1158         if (values_2015 == 0)
1159         {
1160           iterator_counter_35_o--;
1161         }
1162 
1163         sum_35_o += values_2015;
1164       } avg_35_o = sum_35_o / iterator_counter_35_o;
1165       if (age_group == 50)
1166       {
1167         iterator_counter_50_o++;
1168         char add_val_2016[10];
1169         char stg_val_2016[10];
1170         strcpy(add_val_2016, data_set[j].values);
1171         strcpy(stg_val_2016, &add_val_2016[1]);
1172         double values_2016 = atof(stg_val_2016);
1173         if (values_2016 == 0)
1174         {
1175           iterator_counter_50_o--;
1176         }
1177 
1178         sum_50_o += values_2016;
1179       } avg_50_o = sum_50_o / iterator_counter_50_o;
1180       if (age_group == 65)
1181       {
1182         iterator_counter_65_o++;
1183         char add_val_2017[10];
1184         char stg_val_2017[10];
1185         strcpy(add_val_2017, data_set[j].values);
1186         strcpy(stg_val_2017, &add_val_2017[1]);
1187         double values_2017 = atof(stg_val_2017);
1188         if (values_2017 == 0)
1189         {
```

```
1190          iterator_counter_65_o--;
1191        }
1192
1193        sum_65_o += values_2017;
1194      } avg_65_o = sum_65_o / iterator_counter_65_o;
1195    }
1196    for (int j = 128; j < 170; j++) /*  Alberta Age-Group-
    Wise Average Calculator */
1197    {
1198      char add_val_y[10];
1199      char stg_val_y[10];
1200      strcpy(add_val_y, data_set[j].age_group);
1201      strcpy(stg_val_y, &add_val_y[1]);
1202      double age_group = atof(stg_val_y);
1203
1204      if (age_group == 35)
1205      {
1206        iterator_counter_35_a++;
1207        char add_val_2015[10];
1208        char stg_val_2015[10];
1209        strcpy(add_val_2015, data_set[j].values);
1210        strcpy(stg_val_2015, &add_val_2015[1]);
1211        double values_2015 = atof(stg_val_2015);
1212        if (values_2015 == 0)
1213        {
1214          iterator_counter_35_a--;
1215        }
1216
1217        sum_35_a += values_2015;
1218      } avg_35_a = sum_35_a / iterator_counter_35_a;
1219      if (age_group == 50)
1220      {
1221        iterator_counter_50_a++;
1222        char add_val_2016[10];
1223        char stg_val_2016[10];
1224        strcpy(add_val_2016, data_set[j].values);
1225        strcpy(stg_val_2016, &add_val_2016[1]);
1226        double values_2016 = atof(stg_val_2016);
1227        if (values_2016 == 0)
1228        {
1229          iterator_counter_50_a--;
1230        }
1231
1232        sum_50_a += values_2016;
1233      } avg_50_a = sum_50_a / iterator_counter_50_a;
1234      if (age_group == 65)
1235      {
1236        iterator_counter_65_a++;
1237        char add_val_2017[10];
```

```
1238        char stg_val_2017[10];
1239        strcpy(add_val_2017, data_set[j].values);
1240        strcpy(stg_val_2017, &add_val_2017[1]);
1241        double values_2017 = atof(stg_val_2017);
1242        if (values_2017 == 0)
1243        {
1244          iterator_counter_65_a--;
1245        }
1246
1247        sum_65_a += values_2017;
1248      } avg_65_a = sum_65_a / iterator_counter_65_a;
1249    }
1250    for (int j = 170; j < 212; j++) /*  British Columbia Age-
     Group-Wise Average Calculator  */
1251    {
1252      char add_val_y[10];
1253      char stg_val_y[10];
1254      strcpy(add_val_y, data_set[j].age_group);
1255      strcpy(stg_val_y, &add_val_y[1]);
1256      double age_group = atof(stg_val_y);
1257
1258      if (age_group == 35)
1259      {
1260        iterator_counter_35_b++;
1261        char add_val_2015[10];
1262        char stg_val_2015[10];
1263        strcpy(add_val_2015, data_set[j].values);
1264        strcpy(stg_val_2015, &add_val_2015[1]);
1265        double values_2015 = atof(stg_val_2015);
1266        if (values_2015 == 0)
1267        {
1268          iterator_counter_35_b--;
1269        }
1270
1271        sum_35_b += values_2015;
1272      } avg_35_b = sum_35_b / iterator_counter_35_b;
1273      if (age_group == 50)
1274      {
1275        iterator_counter_50_b++;
1276        char add_val_2016[10];
1277        char stg_val_2016[10];
1278        strcpy(add_val_2016, data_set[j].values);
1279        strcpy(stg_val_2016, &add_val_2016[1]);
1280        double values_2016 = atof(stg_val_2016);
1281        if (values_2016 == 0)
1282        {
1283          iterator_counter_50_b--;
1284        }
1285
```

```
1286        sum_50_b += values_2016;
1287      } avg_50_b = sum_50_b / iterator_counter_50_b;
1288      if (age_group == 65)
1289      {
1290        iterator_counter_65_b++;
1291        char add_val_2017[10];
1292        char stg_val_2017[10];
1293        strcpy(add_val_2017, data_set[j].values);
1294        strcpy(stg_val_2017, &add_val_2017[1]);
1295        double values_2017 = atof(stg_val_2017);
1296        if (values_2017 == 0)
1297        {
1298          iterator_counter_65_b--;
1299        }
1300
1301        sum_65_b += values_2017;
1302      } avg_65_b = sum_65_b / iterator_counter_65_b;
1303    }
1304    } age_avg(avg_35_f, avg_50_f, avg_65_f, avg_35_q,
     avg_50_q, avg_65_q, avg_35_o, avg_50_o, avg_65_o, avg_35_a
     , avg_50_a, avg_65_a, avg_35_b, avg_50_b, avg_65_b);
1305
1306  spacer();
1307
1308  /*          Question 2          */printf("
     |-------------------------------------------------------------------------
     Question
     2-------------------------------------------------------------------------
     n");
1309
1310  spacer();
1311
1312  double ProvinceData[4] = {avg_quebec, avg_ontario,
     avg_alberta, avg_british_columbia};
1313
1314  float lowest = 0;
1315  int l_counter = 0;
1316  float highest = 0;
1317  int h_counter = 0;
1318
1319  for (int i = 0; i < 4; i++)
1320  {
1321    if (i == 0)
1322    {
1323      lowest = ProvinceData[i];
1324      l_counter = i;
1325      highest = ProvinceData[i];
1326      h_counter = i;
1327    }
```

```
1328      else
1329      {
1330        if (ProvinceData[i] < lowest)
1331        {
1332          lowest = ProvinceData[i];
1333          l_counter = i;
1334        }
1335        if (ProvinceData[i] > highest)
1336        {
1337          highest = ProvinceData[i];
1338          h_counter = i;
1339        }
1340      }
1341    }
1342
1343    if (l_counter == 0)
1344    {
1345      printf("The Province with the Lowest percentage of
       Diabetes is Quebec\n");
1346    }
1347    if (l_counter == 1)
1348    {
1349      printf("The Province with the Lowest percentage of
       Diabetes is Ontario\n");
1350    }
1351    if (l_counter == 2)
1352    {
1353      printf("The Province with the Lowest percentage of
       Diabetes is Alberta\n");
1354    }
1355    if (l_counter == 3)
1356    {
1357      printf("The Province with the Lowest percentage of
       Diabetes is British Columbia\n");
1358    }
1359
1360    if (h_counter == 0)
1361    {
1362      printf("The Province with the Highest percentage of
       Diabetes is Quebec\n");
1363    }
1364    if (h_counter == 1)
1365    {
1366      printf("The Province with the Highest percentage of
       Diabetes is Ontario\n");
1367    }
1368    if (h_counter == 2)
1369    {
```

```
1370      printf("The Province with the Highest percentage of
          Diabetes is Alberta\n");
1371    }
1372    if (h_counter == 3)
1373    {
1374      printf("The Province with the Highest percentage of
          Diabetes is British Columbia\n");
1375    }
1376
1377    spacer();
1378
1379    /*           Question 3           */printf("
          |-------------------------------------------------------------------------
          Question
          3-------------------------------------------------------------------------
          n");
1380
1381    spacer();
1382
1383    printf("Provinces with a Diabetes percentage above the
          National Average are:\n\n");
1384
1385    for (int i = 0; i < 4; i++)
1386    {
1387      if (ProvinceData[i] > avg_federal)
1388      {
1389        if (i == 0)
1390        {
1391          printf("Qubec\n");
1392        }
1393        if (i == 1)
1394        {
1395          printf("Ontario\n");
1396        }
1397        if (i == 1)
1398        {
1399          printf("Alberta\n");
1400        }
1401        if (i == 1)
1402        {
1403          printf("British Columbia\n");
1404        }
1405
1406
1407      }
1408    }
1409
1410    spacer();
1411
```

```
1412   /*           Question 4           */printf("
        |-------------------------------------------------------------------------
        Question
        4-------------------------------------------------------------------------
        n");
1413
1414   spacer();
1415
1416   lowest = 0, highest = 0;  /*  Re-Initializing lowest &
        highest to 0 */
1417   l_counter = 0, h_counter = 0; /*  Re-Initializing l_counter
        & h_counter to 0  */
1418
1419   for (int i = 0; i < ARRAY_SIZE; i++)
1420     {
1421     char add_val_y[10];
1422     char stg_val_y[10];
1423     strcpy(add_val_y, data_set[i].values);
1424     strcpy(stg_val_y, &add_val_y[1]);
1425     double values = atof(stg_val_y);
1426
1427     if (i == 0)
1428     {
1429       lowest = values;
1430       l_counter = i;
1431       highest = values;
1432       h_counter = i;
1433     }
1434     else
1435     {
1436       if (values < lowest)
1437       {
1438         lowest = values;
1439         l_counter = i;
1440       }
1441       if (values > highest)
1442       {
1443         highest = values;
1444         h_counter = i;
1445       }
1446     }
1447   } printf("The Province with the highest & lowest percentage
        of diabetes in a year is British Columbia & Ontario in
       the year's %s & %s\n", data_set[h_counter].year, data_set[
       l_counter].year);
1448
1449   spacer();
1450 }
1451
```

```c
/*  Defining all subsequent UDF's utilized in the program */

void credits(void)
{
  spacer();
  printf("                      Toronto Metropolitan University\n");
  printf("*******************************\n");
  printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
  sub_spacer();
  printf("---------------------------------------------------------------------------\n");
  printf("                 CPS 188 Term Project | 2023 Source Code\n");
  printf("---------------------------------------------------------------------------\n");
  sub_spacer();
  printf("                               ------------\n");
  printf("-------------------------------|Group - 40|-------------------------------\n");
  printf("                               ------------\n");
  sub_spacer();
  printf("Copyright (c) 2023 Sayeed Ahamad, Qurrat-Ul-Ain, Tre Spencer, Nourhan Antar\n");
  spacer();
  printf("/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\n");
  spacer();
}

FILE* file_o(void)   /*  UDF for file opening    */
{
    char temp[100];
    FILE* file; /*  Initializing file storage variable  */
    file = fopen("statscan_diabetes.csv", "r"); /*  Defining the address path of file to be opened and read    */
    if (NULL == file)   /*  Fail Safe mechanism for program file opening    */
    {
        printf("Error: File Open cannot proceed\n");
        exit;   /*  Program exits and ends if the fail-safe breaks  */
    }
```

```
1485     fscanf(file, "%s", temp);  /*  Scanning the defined file
         and storing it at address of Var(temp)    */
1486     return file;    /*  Returning the file type variable
         */
1487 }
1488
1489 void file_c(FILE* file) /*  UDF for file closing    */
1490 {
1491     fclose(file);   /*  Closing the already opened file */
1492 }
1493
1494 void spacer(void) /*  UDF for creating NULL newline spaces
         for better tabular and visual output */
1495 {
1496     for (int i = 0; i < SPACERS; i++)
1497     {
1498         printf("\n");
1499     }
1500
1501 }
1502
1503 void sub_spacer(void) /*  UDF for creating NULL newline
         spaces for better tabular and visual output */
1504 {
1505     for (int i = 0; i < SUB_SPACERS; i++)
1506     {
1507         printf("\n");
1508     }
1509
1510 }
1511
1512 void avg_province(void) /*  UDF for Province-Wise Averages
         Header */
1513 {
1514   printf("|--------------------Province-Wise Averages
         --------------------|\n");
1515 }
1516
1517 void avg_year(void) /*  UDF for Year-Wise Averages Header */
1518 {
1519   printf("|--------------------Year-Wise Averages
         --------------------|\n");
1520 }
1521
1522 void year_avg(double f_2015, double f_2016, double f_2017,
         double f_2018, double f_2019, double f_2020, double f_2021
         , double q_2015, double q_2016, double q_2017, double
         q_2018, double q_2019, double q_2020, double q_2021,
         double o_2015, double o_2016, double o_2017, double o_2018
```

```
    , double o_2019, double o_2020, double o_2021, double
    a_2015, double a_2016, double a_2017, double a_2018,
    double a_2019, double a_2020, double a_2021, double b_2015
    , double b_2016, double b_2017, double b_2018, double
    b_2019, double b_2020, double b_2021)  /*  UDF to print
    Year-Wise Averages in a tabular form */
1523 {
1524   printf("| Year  |   Canada    |   Quebec    |   Ontario   |
       Alberta   |   British Columbia    |\n");
1525   printf("
     |-------------------------------------------------------------------------
     n");
1526   printf("| 2015  |   %.3lf   |   %.3lf   |   %.3lf   |   %.3
     lf   |     %.3lf     |\n", f_2015, q_2015, o_2015, a_2015,
      b_2015);
1527   printf("| 2016  |   %.3lf   |   %.3lf   |   %.3lf   |   %.3
     lf   |     %.3lf     |\n", f_2016, q_2016, o_2016, a_2016,
      b_2016);
1528   printf("| 2017  |   %.3lf   |   %.3lf   |   %.3lf   |   %.3
     lf   |     %.3lf     |\n", f_2017, q_2017, o_2017, a_2017,
      b_2017);
1529   printf("| 2018  |   %.3lf   |   %.3lf   |   %.3lf   |   %.3
     lf   |     %.3lf     |\n", f_2018, q_2018, o_2018, a_2018,
      b_2018);
1530   printf("| 2019  |   %.3lf   |   %.3lf   |   %.3lf   |   %.3
     lf   |     %.3lf     |\n", f_2019, q_2019, o_2019, a_2019,
      b_2019);
1531   printf("| 2020  |   %.3lf   |   %.3lf   |   %.3lf   |   %.3
     lf   |     %.3lf     |\n", f_2020, q_2020, o_2020, a_2020,
      b_2020);
1532   printf("| 2021  |   %.3lf   |   %.3lf   |   %.3lf   |   %.3
     lf   |     %.3lf     |\n", f_2021, q_2021, o_2021, a_2021,
      b_2021);
1533 }
1534
1535 void avg_age(void)  /*  UDF for Age-Group-Wise Averages
    Header  */
1536 {
1537   printf("|--------------------Age-Group-Wise Averages
    --------------------|\n");
1538 }
1539
1540 void age_avg(double f_35, double f_50, double f_65, double
    q_35, double q_50, double q_65, double o_35, double o_50,
    double o_65, double a_35, double a_50, double a_65, double
     b_35, double b_50, double b_65) /*  UDF to print Age-
    Group-Wise Averages in a tabular form  */
1541 {
```

```
1542    printf("| Age-Group |   Canada    |   Quebec    |   Ontario
            |   Alberta   |   British Columbia    |\n");
1543    printf("
          |-----------------------------------------------------------------------------
          n");
1544    printf("| 35-49 |   %.3lf   |   %.3lf   |   %.3lf   |   %.3
          lf   |     %.3lf     |\n", f_35, q_35, o_35, a_35, b_35);
1545    printf("| 50-64 |   %.3lf   |   %.3lf   |   %.3lf   |   %.3
          lf   |     %.3lf     |\n", f_50, q_50, o_50, a_50, b_50);
1546    printf("| 65+ |   %.3lf   |   %.3lf   |   %.3lf   |   %.3lf
            |     %.3lf     |\n", f_65, q_65, o_65, a_65, b_65);
1547 }
```

Listing A.1: *CPS 188 Term Project Source Code*

B GNUplot Scripts

B.1 Question 5

```
1
2
3  set title "Diabetes Percentages in Canada (2015-2021)"
4  set xlabel "Year"
5  set ylabel "Diabetes Percentage"
6  set xtics 1
7  set key outside right
8
9  set style line 1 lw 2 pt 7
10 set style line 2 lw 2 pt 5
11 set style line 3 lw 2 pt 9
12 set style line 4 lw 2 pt 13
13 set style line 5 lw 2 pt 11
14
15 plot "data4a.txt" using 1:2 with lines linestyle 1 linecolor
       rgb "#E41A1C" title "Canada ex. territories", \
16       "data4a.txt" using 1:3 with lines linestyle 2 linecolor
       rgb "#377EB8" title "British Columbia", \
17       "data4a.txt" using 1:4 with lines linestyle 3 linecolor
       rgb "#4DAF4A" title "Alberta", \
18       "data4a.txt" using 1:5 with lines linestyle 4 linecolor
       rgb "#984EA3" title "Ontario", \
19       "data4a.txt" using 1:6 with lines linestyle 5 linecolor
       rgb "#FF7F00" title "Quebec"
```

Listing B.1: *GNUplot Source Code*

B.2 Question 6

```
1
2
3  set title "Average Percentages of Diabetes Among Age Groups"
4  set xlabel "Age Groups"
5  set ylabel "Diabetes Percentage"
6  set style data histograms
7  set style fill solid 1.0 border -1
8  set boxwidth 0.7 relative
9  set yrange [0:20]
10 set ytics 0,2,20
11 plot 'q6.txt' using 2:xtic(1) with histogram title "Diabetes
       Percentage"
```

Listing B.2: *GNUplot Source Code*