Credit Card II Project, Fall 2025

Tong Yi

Contents

1 Task A: Find the maximum monthly total across all categories; draw a chart for a selected category **1**
1.1 Example of Task A . . . 2
1.2 Hints for Task A: Find the Maximum Value of a Two-Dimensional Array . . . 4
1.3 Key Difference between Medal Counting Program and Credit Card Project . . . 7
1.4 Calculate Number of Asterisks . . . 7
1.5 Steps for Task A . . . 8
1.6 Submission of Task A . . . 10

2 Task B: Print Monthly Category Table **10**
2.1 Sample Input / Output of Task B . . . 11
2.2 Hints for Task B . . . 11

3 Task C: Generate a CSV File of Monthly Category **13**
3.1 Hints for Task C . . . 15
3.2 Use the Above Source code . . . 18

Warning:

1. This is copyrighted materials; you are not allowed to upload to the Internet.
2. Our project is different from similar products in Internet.
   (a) Ask help only from teaching staff of this course.
   (b) Use solutions from artificial intelligence like ChatGPT or online tutoring websites like, but not limited to, chegg.com, violates academic integrity and is not allowed.

1 Task A: Find the maximum monthly total across all categories; draw a chart for a selected category

1. Create a file named `monthly_category_total_chart.cpp`. The input file is a yearly credit card report containing up to 12 months and no more than 20 categories.
2. In the program, prompt the user to enter the name of an input file.
3. Read the file.

4. Identify and sort all categories alphabetically.
5. Find the maximum monthly total across ***all*** categories.
6. Display all categories alphabetically and prompt the user to choose one.
7. Display a chart of asterisks representing the monthly totals of the selected category.

1.1 Example of Task A

Create a file named **spending.csv** with the following contents. On Mac/Linux, you can use VS Code, TextEdit, Vim, or Emacs. On Windows, you can use VS Code, Notepad, Notepad++, or Vim. The file `spending.csv` contains `Date`, `Description`, `Category`, and `Amount`.

```
Date,Description,Category,Amount
01/16/2024,Con Edison,Utilities,91.35
02/14/2024,National Grid,Utilities,32.75
03/17/2024,Macy's,Shopping,109.3
03/19/2024,Taxi,Travel,71.37
03/24/2024,Marshalls,Shopping,98.76
03/30/2024,National Grid,Utilities,10.93
03/31/2024,MTA,Travel,20.38
05/25/2024,Macy's,Shopping,32.87
06/15/2024,Macy's,Shopping,55.49
08/21/2024,Banana Republic,Shopping,59.85
08/25/2024,National Grid,Utilities,27.16
```

The file has three categories, listed in alphabetical order.

```
Shopping
Travel
Utilities
```

Here is a monthly total for each category based on the above data.

Month	Shopping	Travel	Utilities
1	0	0	91.35
2	0	0	32.75
3	109.3 + 98.76 = 208.06	71.37 + 20.38 = 91.75	10.93
4	0	0	0
5	32.87	0	0
6	55.49	0	0
7	0	0	0
8	59.85	0	27.16
9	0	0	0
10	0	0	0
11	0	0	0
12	0	0	0

Explanation:

1. On 3/17/2024, we spent 109.3 on shopping, and on 3/24/2024, another 98.76 on shopping. Therefore, the total spent on shopping in March 2024 is 109.3 + 98.76 = 208.06.

2. Similarly, on 3/19/2024, we spent 71.37 on travel and on 3/31/2024, another 20.38 on travel. Therefore, the total spent on travel in March 2024 is 71.37 + 20.38 = 91.75.

March 2024 Spending Summary

Shopping

Date	Amount
03/17/2024	109.30
03/24/2024	98.76
Total	208.06

Travel

Date	Amount
03/19/2024	71.37
03/31/2024	20.38
Total	91.75

Utilities

Date	Amount
03/30/2024	10.93
Total	10.93

Here is a sample output for the above `spending.csv`.

```
Enter a file name: spending.csv (with return key)
select one of the following categories
0.Shopping
1.Travel
2.Utilities
choose a number in [0, 2]: 0 (with return key)
max monthly total across all categories = 208.06
MONTH  Shopping TOTAL
Jan              0.00
Feb              0.00
Mar            208.06 ****************************************
Apr              0.00
May             32.87 ******
Jun             55.49 **********
Jul              0.00
Aug             59.85 ***********
Sep              0.00
Oct              0.00
Nov              0.00
Dec              0.00
```

another sample input/output:

```
Enter a file name: spending.csv (with return key)
select one of the following categories
0.Shopping
1.Travel
2.Utilities
choose a number in [0, 2]: 1 (with return key)
max monthly total across all categories = 208.06
MONTH    Travel TOTAL
Jan              0.00
Feb              0.00
```

```
Mar              91.75 *****************
Apr               0.00
May               0.00
Jun               0.00
Jul               0.00
Aug               0.00
Sep               0.00
Oct               0.00
Nov               0.00
Dec               0.00
```

yet another sample input/output:

```
Enter a file name: spending.csv (with return key)
select one of the following categories
0.Shopping
1.Travel
2.Utilities
choose a number in [0, 2]: 2 (with return key)
max monthly total across all categories = 208.06
Jan              91.35 *****************
Feb              32.75 ******
Mar              10.93 **
Apr               0.00
May               0.00
Jun               0.00
Jul               0.00
Aug              27.16 *****
Sep               0.00
Oct               0.00
Nov               0.00
Dec               0.00
```

1.2 Hints for Task A: Find the Maximum Value of a Two-Dimensional Array

Consider the medal-count CSV file **medals_original.csv** from the 2014 Winter Olympic skating competitions from Section 6.6 in the textbook.

Country	Gold	Silver	Bronze
Canada	0	3	0
Italy	0	0	1
Germany	0	0	1
Japan	1	0	0
Kazakhstan	0	0	1
Russia	3	1	1
South Korea	0	1	0
United States	1	0	1

Here is a two-dimensional array `counts` using the above data. We did not need to sort the countries names in this example. For example, `counts[0][1]`, where row index is 0 and column index is 1 is 3.

counts	~~Gold~~ 0	~~Silver~~ 1	~~Bronze~~ 2
~~Canada~~ 0	0	3	0
~~Italy~~ 1	0	0	1
~~Germany~~ 2	0	0	1
~~Japan~~ 3	1	0	0
~~Kazakhstan~~ 4	0	0	1
~~Russia~~ 5	3	1	1
~~South Korea~~ 6	0	1	0
~~United States~~ 7	1	0	1

Find out the maximum element in two dimensional array `counts`, as shown in `https://onlinegdb.com/QtmcJBfre`. Source code can be downloaded from **max_number_medals.cpp**.

Warning: onlinegdb does not recognize files with suffix csv, so we need to rename the input file as `medals_original.txt`.

```
1  //code link: https://onlinegdb.com/QtmcJBfre
2
3  #include <iostream>
4  #include <string>
5  #include <fstream> //std::ifstream
6
7  //Suppose medals_original.csv has the following contents,
8  //Country,Gold,Silver,Bronze
9  //Canada,0,3,0
10 //Italy,0,0,1
11 //Germany,0,0,1
12 //Japan,1,0,0
13 //Kazakhstan,0,0,1
14 //Russia,3,1,1
15 //South Korea,0,1,0
16 //United States,1,0,1
17
18 //Find out the maximum entry in the above file.
19
20 //Sample input/output:
21 //Enter a csv file name with countries and medals (gold, silver, and bronze):
      medals_original.csv
22 //maximum number of medals: 3
23 int main() {
24     //Enter a csv file name to read
25     std::cout << "Enter a csv file name with countries and medals (gold, silver, and
      bronze): ";
26     std::string fileName;
27     std::cin >> fileName;
28
```

```
    //create std::ifstream object fin to read a file with fileName
    std::ifstream fin(fileName);

    //fin cannot open the associated file
    if (fin.fail()) {
       std::cerr << fileName << " cannot be opened" << std::endl;
       exit(1);
    }

    //read the first line, which are column headers without further processing
    std::string columnHeaders;
    getline(fin, columnHeaders);

    //The next line of data contains a country's name,
    //followed by three integers:
    //numGold, numSilver, and numBronze, respectively.
    std::string country;
    std::string numGoldStr;
    int numGold;
    std::string numSilverStr;
    int numSilver;
    std::string numBronzeStr;
    int numBronze;

    const int COUNTRY_CAPACITY = 100; //maximum number of countries
    const int MEDALS = 3; //types of medals
    int counts[COUNTRY_CAPACITY][MEDALS];
    std::string countryNames[COUNTRY_CAPACITY];

    int numCountries = 0;

    //Since the file is in CSV format,
    //we need to use
    //istream& getline(istream& is, string& str, char delim);
    while (getline(fin, country, ',')) { //search for a string before delimiter , save
   in variable country
        countryNames[numCountries] = country;

        getline(fin, numGoldStr, ','); //number of gold medals is followed by ','
        numGold = stoi(numGoldStr);
        counts[numCountries][0] = numGold;

        getline(fin, numSilverStr, ','); //number of silver medals is followed by ','
        numSilver = stoi(numSilverStr);
        counts[numCountries][1] = numSilver;

```

```
74          getline(fin, numBronzeStr); //number of bronze medals is followed by '\n', no
    need to specify delimiter.
75          numBronze = stoi(numBronzeStr);
76          counts[numCountries][2] = numBronze;
77
78          numCountries++; //number of countries is increased by 1
79      }
80      fin.close();
81
82      //The medals program is different from credit card project.
83      //In medals, there are only three types of medals,
84      //while in a credit card, the types of categories can differ from one file to
    another,
85      //so we need to read the file twice in credit card project,
86      //In the first round, save the categories in alphabetic order to an array.
87      //In the second round, populate the data of two-dimensional array.
88
89      int max = counts[0][0];
90      for (int i = 0; i < numCountries; i++) { //i is row index
91          for (int j = 0; j < MEDALS; j++) { //j is column index, there are MEDALS many
    columns in each row
92              if (counts[i][j] > max) {
93                  max = counts[i][j];
94              }
95          }
96      }
97
98      std::cout << "maximum number of medals: " << max << std::endl;
99      return 0;
100 }
```

1.3 Key Difference between Medal Counting Program and Credit Card Project

In a medal-counting program, the categories – Gold, Silver, and Bronze – are fixed and known in advance. In contrast, a credit card project must handle dynamic categories.

Different credit card reports may have different number and types of categories. So the indices of a category in the categories array can be different as well.

For example, in one credit card report with categories: Shopping, Traveling, and Utilities, the categories array has three elements and Shopping is at index 0.

In another credit card report with categories: Entertainment, Shopping, Traveling, and Utilities, the categories array has four elements and Shopping is at index 1.

1.4 Calculate Number of Asterisks

1. The number of asterisks representing the maximum monthly total across ***all*** categories is `MAX_NUM_ASTS`, which is assumed to be 40.

2. In the above `spending.csv`, the maximum monthly total across all categories is 208.06 (as seen in Shopping in March) and is represented by 40 asterisks.

3. For any other monthly total, the number of asterisks is calculated as

   `(monthly_total / max_monthly_total_across_all_categories) * MAX_NUM_ASTS`.

   The decimal part should be truncated.

   Example: In the above `spending.csv`, the maximum monthly total across all category is 208.06. Calculate the number of asterisks for the Shopping category in May 2024 as follows.

   (a) The total spending in Shopping is 32.87.

   (b) The number of asterisks is calculated by (32.87 / 208.06) * 40 = 6.31.

   (c) After **truncating** decimals, this results in 6 asterisks.

1.5 Steps for Task A

1. Enter the name of the yearly credit card report CSV file.

2. Create a `std::ifstream` object, `fin`, that is associated with the input CSV file.

3. If `fin` fails to open the input file, prompt an error message and exit to the operating system with error code 1.

4. Read the input file, identify and save the categories in the file in alphabetical order in an array of strings called `categories`.

   For example, the file `spending.csv` in 1.1 has three categories: "Shopping", "Travel", and "Utilities". The contents of `categories` array are as follows. The first row represents indices.

0	1	2
Shopping	Travel	Utilities

5. Use a two-dimensional array `categoryTotal` of type `double`, where the first dimension (rows) represents months and the second dimension (columns) represents categories in alphabetical order.

   We assume the input file is a yearly credit card report containing up to 12 months and no more than 20 categories. The two-dimensional array can therefore be declared as follows:

```
const int NUM_MONTHS = 12;
const int MAX_NUM_CATEGORIES = 20;
double categoryTotal[NUM_MONTHS][MAX_NUM_CATEGORIES];
```

6. Initialize each element of array `categoryTotal` to be zero.

7. Read the file again. After reading it once to identify the category information, the file's read position reaches the end. To rewind the file pointer to the beginning without closing and reopening the file, use the following code.

```
fin.clear();
fin.seekg(0); //return the beginning of the file
```

For each record, do the following:

(a) Extract the month information.

(b) Find out the corresponding index of that category in the `categories` array. Save it in the variable `categoryIdx`.

This step is very important. We need to map a category name to its corresponding index in the `categories` array.

(c) Increase the value in the `categoryTotal` at `(month-1)` row index and `categoryIdx` column index by the corresponding amount in the record.

We subtract 1 from the `month` value to get the row index because `month` starts from 1 (January), however, row index starts from 0.

For example, given the following record,

```
01/16/2024,Con Edison,Utilities,91.35
```

i. The month value of date "01/16/2024" is 1 (January), which corresponds to a row index 0.

ii. The category is `Utilities`, which is indexed at 2 – the third element – in array `categories`, assuming the order is `Shopping`, `Travel`, and `Utilities`.

iii. Increase `categories[0][2]` by 91.35, the amount of the record. The entry in the `categories` array that needs to be increased is shown below.

```
MON        Shopping          Travel        Utilities
-------------------------------------------------------
Jan            0.00            0.00           91.35
Feb            0.00            0.00           32.75
Mar          208.06           91.75           10.93
Apr            0.00            0.00            0.00
May           32.87            0.00            0.00
Jun           55.49            0.00            0.00
Jul            0.00            0.00            0.00
Aug           59.85            0.00           27.16
Sep            0.00            0.00            0.00
Oct            0.00            0.00            0.00
Nov            0.00            0.00            0.00
Dec            0.00            0.00            0.00
```

Given another record,

```
03/17/2024,Macy's,Shopping,109.3
```

iv. The month value of date "03/17/2024" is 3 (March), which corresponds to a row index 2.

v. The category is `Shopping`, which is indexed at 0 – the first element – in array `categories`, assuming the order is `Shopping`, `Travel`, and `Utilities`.

vi. Increase `categories[2][0]` by 109.3, the amount of the record. Then `categories[2][0]` is changed from 0 to 109.3.

Given yet another record,

```
03/24/2024,Marshalls,Shopping,98.76
```

vii. The month value of date "03/24/2024" is 3 (March), which corresponds to a row index 2.

viii. The category is `Shopping`, which is indexed at 0 – the first element – in array `categories`, assuming the order is `Shopping`, `Travel`, and `Utilities`.

ix. Increase `categories[2][0]` by 98.76, the amount of the record. Then `categories[2][0]` is changed from 109.3 to 109.3 + 98.76 = 208.06.

```
MON        Shopping          Travel       Utilities
-------------------------------------------------------------
Jan            0.00            0.00           91.35
Feb            0.00            0.00           32.75
Mar          208.06           91.75            10.93
Apr            0.00            0.00            0.00
May           32.87            0.00            0.00
Jun           55.49            0.00            0.00
Jul            0.00            0.00            0.00
Aug           59.85            0.00           27.16
Sep            0.00            0.00            0.00
Oct            0.00            0.00            0.00
Nov            0.00            0.00            0.00
Dec            0.00            0.00            0.00
```

8. Find out the maximum monthly total across all categories.

9. Display the entries of array `categories` in alphabetical order, labeling the first item as 0. Prompt the user to select a category, and display its monthly totals.

10. For each monthly total, calculate the number of asterisks according to the rule in Section 1.3, and print them.

11. Close the input file by calling the `close` method of `std::ifstream` object `fin`.

1.6 Submission of Task A

Submit `monthly_category_total_chart.cpp` to gradescope. Note that the grading script uses random double numbers to test. As a result, your output will be different in each running.

2 Task B: Print Monthly Category Table

Print a table with the following contents.

1. For each row, display the **monthly total per category**, then the **total for that month**.

2. After the monthly data, print the **annual total for each category** and the **overall annual total**.

3. Finally, print **each category's annual percentage** relative to the overall annual total.

2.1 Sample Input / Output of Task B

For `spending.csv` as in Section 1.1, the print out should be as follows.

```
Enter a csv file name with date, category, cost: spending.csv (with return key)
MON       Shopping         Travel      Utilities          TOTAL
-----------------------------------------------------------------
Jan           0.00           0.00          91.35          91.35
Feb           0.00           0.00          32.75          32.75
Mar         208.06          91.75          10.93         310.74
Apr           0.00           0.00           0.00           0.00
May          32.87           0.00           0.00          32.87
Jun          55.49           0.00           0.00          55.49
Jul           0.00           0.00           0.00           0.00
Aug          59.85           0.00          27.16          87.01
Sep           0.00           0.00           0.00           0.00
Oct           0.00           0.00           0.00           0.00
Nov           0.00           0.00           0.00           0.00
Dec           0.00           0.00           0.00           0.00
=================================================================
            356.27          91.75         162.19         610.21
             58.38%         15.04%         26.58%
```

Explanation:

1. January Spending is shown as follows. No spending in Shopping and Traveling, but 91.35 for Utilities, so the monthly total is 91.35.

```
MON       Shopping         Travel      Utilities          TOTAL
-----------------------------------------------------------------
Jan           0.00           0.00          91.35          91.35
```

2. In March, spend 208.06 in Shopping, 91.75 in Traveling, and 10.93 in Utilities, so the monthly total is 208.06 + 91.75 + 10.93 = 310.74.

3. In the whole year, spend 356.27 in Shopping, 91.75 in Traveling, and 162.19 in Utilities. The annual total is 356.27 + 91.75 + 162.19 = 610.21.

4. In the whole year, spend 356.27 / 610.21 = 58.38% in Shopping, 91.75 / 610.21 = 15.04% in Traveling, and 162.19 / 610.21 = 26.58% in Utilities.

2.2 Hints for Task B

1. Need to include libraries `iostream` and `iomanip`. The `iomanip` library contains input/output manipulators that are useful for formatting the program's output, such as aligning text or controlling decimal precision.

2. To print the following header line, where a space character `' '` is displayed as ␣.

```
MON␣␣␣␣␣␣␣Shopping␣␣␣␣␣␣␣␣␣Travel␣␣␣␣␣␣Utilities␣␣␣␣␣␣␣␣␣␣TOTAL
```

In the following code, we use `std::setw(15)` because we assume category names will have a maximum length of 14 characters (including spaces). This ensures at least one space of padding for every category name.

```
std::cout << std::left << "MON"; //"MON" for "MONTH",
//use "MON" instead of "MONTH" to be consistent with 3-letter month name
   abbreviation

//Display elements of array categories.
//std::right means the print out is right aligned.
//std::setw(15) means categories[j] is displayed within a field of at least 15
   characters.
//Warning: need to declare and initialize category_size properly.
for (int j = 0; j < category_size; j++) {
    std::cout << std::right << std::setw(15) << categories[j];
}

std::cout << std::right << std::setw(15) << "TOTAL" << "\n";
```

(a) `std::right`: This manipulator sets the justification of the output within the specified field width. It causes the value to be printed at the right edge of the field, with any extra space added to the left.

(b) `std::setw(15)`: The setw (set width) manipulator sets the minimum field width for the next output operation to 15 characters. If the value of `categories[j]` is smaller than this width, the field will be padded with spaces. This setting only applies to the next value printed to the stream.

So, there are 8 letters in `"Shopping"`. With `std::setw(15)`, there are 15 - 8 = 7 spaces before `"Shopping"`.

Similarly, there are 9 letters in `"Traveling"`. With `std::setw(15)`, there are 15 - 9 = 6 spaces before `"Traveling"`.

3. To display the contents of two-dimensional array `categoryTotal`, where the row index represents a month, and the column index represents a category.

```
const in NUM_MONTHS = 12;
std::string month_names[] = {"Jan", "Feb", "Mar",
   "Apr", "May", "Jun", "Jul", "Aug", "Sep",
   "Oct", "Nov", "Dec"};

//TODO: Declare variable to hold the annual category total.
//TODO: Declare variable to hold the annual total.

double monthly_total;
for (int i = 0; i < NUM_MONTHS; i++) {
    std::cout << std::left << std::setw(3) << month_names[i];
    monthly_total = /* what is the initial value for each month */;
    for (int j = 0; j < category_size; j++) {
```

```
        std::cout << std::right << std::setw(15) << std::fixed << std::
    setprecision(2) << categoryTotal[i][j];

        //TODO: update monthly_total.

        //TODO: update annual category total for current category.
    }

    //TODO: update annual total
    //TODO: display monthly_total
}

//TODO: display annual category total and annual total
//TODO: calculate and display each category's annual percentage
```

In the following code, `categoryTotal[i][j]` is displayed in fixed-point notation, right-aligned, with a width of 15 characters (including digits, the decimal point, and possible spaces padded to the left) and two decimal places.

```
std::cout << std::right << std::setw(15) << std::fixed << std::setprecision(2) <<
    categoryTotal[i][j];
```

The following explanation comes from chatgpt.

(a) `std::right` aligns the output text to the right within its field width (set by setw).

(b) `std::setw(15)` sets the field width to 15 characters. The next value printed (categoryTotal[i][j]) will occupy 15 character spaces. If the number is shorter than 15 characters, spaces will be added on the left (because of `std::right`).

(c) `std::fixed` forces the number to be printed in fixed-point notation rather than scientific notation.

Example:

Without fixed: 1.23e+03

With fixed: 1230.00

(d) `std::setprecision(2)` sets the number of digits after the decimal point to 2.

Example:

3.14159 → 3.14

5 → 5.00

3 Task C: Generate a CSV File of Monthly Category

All the step will be same as Task B, the only difference is not output to the screen, but write to a csv file.

```
Enter a csv file name with date, category, cost: spending.csv
enter the name of generated csv file: monthlyCategory.csv
```

The generated `monthlyCategory.csv` is as follows.

```
MON,Shopping,Travel,Utilities,TOTAL
Jan,0.00,0.00,91.35,91.35
Feb,0.00,0.00,32.75,32.75
Mar,208.06,91.75,10.93,310.74
Apr,0.00,0.00,0.00,0.00
May,32.87,0.00,0.00,32.87
Jun,55.49,0.00,0.00,55.49
Jul,0.00,0.00,0.00,0.00
Aug,59.85,0.00,27.16,87.01
Sep,0.00,0.00,0.00,0.00
Oct,0.00,0.00,0.00,0.00
Nov,0.00,0.00,0.00,0.00
Dec,0.00,0.00,0.00,0.00
,356.27,91.75,162.19,610.21
,58.38%,15.04%,26.58%,
```

Open `monthlyCategory.csv` in Excel in Mac / Windows or Numbers in Mac and it looks like as follows.

	A	B	C	D	E
1	MON	Shopping	Travel	Utilities	TOTAL
2	Jan	0	0	91.35	91.35
3	Feb	0	0	32.75	32.75
4	Mar	208.06	91.75	10.93	310.74
5	Apr	0	0	0	0
6	May	32.87	0	0	32.87
7	Jun	55.49	0	0	55.49
8	Jul	0	0	0	0
9	Aug	59.85	0	27.16	87.01
10	Sep	0	0	0	0
11	Oct	0	0	0	0
12	Nov	0	0	0	0
13	Dec	0	0	0	0
14		356.27	91.75	162.19	610.21
15		58.38%	15.04%	26.58%	

3.1 Hints for Task C

Given medals by countries **medals_original.csv** as in 1.2, generate a CSV file with information of Medal Count By Country and Medals Totals For All Countries.

The onlinegdb code link is `https://onlinegdb.com/8rU18e-SS5`. Warning: onlinegdb does not recognize files with suffix csv, so we need to rename the input file as `medals_original.txt` and the output file as `medals_tally.txt`.

Source code can be downloaded from **medals_to_csv.cpp**.

```
//code link: https://onlinegdb.com/8rU18e-SS5

#include <iostream>
#include <string>
#include <fstream> //std::ifstream

//Suppose medals_original.csv has the following contents,
//Country,Gold,Silver,Bronze
//Canada,0,3,0
//Italy,0,0,1
//Germany,0,0,1
//Japan,1,0,0
//Kazakhstan,0,0,1
//Russia,3,1,1
//South Korea,0,1,0
//United States,1,0,1

//Need to provide medals_original.csv to run this program.
//Sample input/output:
//Enter a csv file name with countries and medals (gold, silver, and bronze):
   medals_original.csv
//Enter an output file name: medals_tally.csv

//Generate medals_tally.csv with the following contents.
//Country,Gold,Silver,Bronze,Medal Count By Country
//Canada,0,3,0,3
//Italy,0,0,1,1
//Germany,0,0,1,1
//Japan,1,0,0,1
//Kazakhstan,0,0,1,1
//Russia,3,1,1,5
//South Korea,0,1,0,1
//United States,1,0,1,2
//Medals Totals For All Countries,5,5,5,15
int main() {
    //Enter a csv file name to read
    std::cout << "Enter a csv file name with countries and medals (gold, silver, and
   bronze): ";
```

```
37      std::string fileName;
38      std::cin >> fileName;
39
40      //create std::ifstream object fin to read a file with fileName
41      std::ifstream fin(fileName);
42
43      //fin cannot open the associated file
44      if (fin.fail()) {
45         std::cerr << fileName << " cannot be opened" << std::endl;
46         exit(1);
47      }
48
49      //read the first line, which are column headers without further processing
50      std::string columnHeaders;
51      getline(fin, columnHeaders);
52
53      //The next line of data contains a country's name,
54      //followed by three integers:
55      //numGold, numSilver, and numBronze, respectively.
56      std::string country;
57      std::string numGoldStr;
58      int numGold;
59      std::string numSilverStr;
60      int numSilver;
61      std::string numBronzeStr;
62      int numBronze;
63
64      const int COUNTRY_CAPACITY = 100; //maximum number of countries
65      const int MEDALS = 3; //types of medals
66      int counts[COUNTRY_CAPACITY][MEDALS];
67      std::string countryNames[COUNTRY_CAPACITY];
68
69      int numCountries = 0;
70
71      //Since the file is in CSV format,
72      //we need to use
73      //istream& getline(istream& is, string& str, char delim);
74      while (getline(fin, country, ',')) { //search for a string before delimiter , save
    in variable country
75          countryNames[numCountries] = country;
76
77          getline(fin, numGoldStr, ','); //number of gold medals is followed by ','
78          numGold = stoi(numGoldStr);
79          counts[numCountries][0] = numGold;
80
81          getline(fin, numSilverStr, ','); //number of silver medals is followed by ','
```

```
82          numSilver = stoi(numSilverStr);
83          counts[numCountries][1] = numSilver;
84
85          getline(fin, numBronzeStr); //number of bronze medals is followed by '\n', no
    need to specify delimiter.
86          numBronze = stoi(numBronzeStr);
87          counts[numCountries][2] = numBronze;
88
89          numCountries++; //number of countries is increased by 1
90      }
91      fin.close();
92
93      //The medals program is different from credit card project.
94      //In medals, there are only three types of medals,
95      //while in a credit card, the types of categories can differ from one file to
    another,
96      //so we need to read the file twice in credit card project,
97      //In the first round, save the categories in alphabetic order to an array.
98      //In the second round, populate the data of two-dimensional array.
99
100     std::cout << "Enter an output file name: ";
101     std::string outputFileName;
102     std::cin >> outputFileName;
103
104     std::ofstream fout(outputFileName);
105     if (fout.fail()) {
106        std::cerr << outputFileName << " cannot be opened." << std::endl;
107        exit(2);
108     }
109
110     //print "Country,Gold,Silver,Bronze,Medal Count By Country\n" to fout
111     //This is column header of the output CSV file
112     fout << "Country,";
113     std::string medal_names[] = {"Gold", "Silver", "Bronze"};
114     for (int j = 0; j < MEDALS; j++) {
115         fout << medal_names[j] << ',';
116     }
117     fout << "Medal Count By Country\n";
118
119     int medalCountByCountry; //Medal Count by Country
120     int total = 0;
121     for (int i = 0; i < numCountries; i++) { //i is row index
122         fout << countryNames[i] << ','; //',' can be written as ","
123         medalCountByCountry = 0;
124         for (int j = 0; j < MEDALS; j++) { //j is column index, there are MEDALS many
    columns in each row
```

```
            fout << counts[i][j] << ",";
            medalCountByCountry += counts[i][j];
        }
        fout << medalCountByCountry << "\n"; //finish the row
        total += medalCountByCountry;
    }

    //Calculate gold-, silver-, and bronze- totals for all countries
    int medalTotalAllCountries[MEDALS];
    for (int j = 0; j < MEDALS; j++) {
        medalTotalAllCountries[j] = 0;

        for (int i = 0; i < numCountries; i++) {
            medalTotalAllCountries[j] += counts[i][j];
        }
    }

    fout << "Medals Totals For All Countries,"; //no entry before the first entry
    for (int j = 0; j < MEDALS; j++) {
        fout << medalTotalAllCountries[j] << ",";
    }
    fout << total; //no need to have a new line, end of file

    fout.close();
    return 0;
}
```

3.2 Use the Above Source code

You can delete the file `medals_tally.txt` to test the code to generate a new file. But do NOT delete `medals_original.txt`, otherwise, you need to open the code link in another tab to test the code.

```
Enter a csv file name with countries and medals (gold, silver, and bronze):
   medals_original.txt
Enter an output file name: medals_tally.txt
```

Afterwards, highlight `medals_tally.txt` and choose download. Rename the file to `medals_tally.csv` so that Excel or Numbers can recognize the file format correctly.

If you open `medals_tally.csv` with Excel, this is what you will get.

	A	B	C	D	E	F
1	Country	Gold	Silver	Bronze	Medal Count By Country	
2	Canada	0	3	0	3	
3	Italy	0	0	1	1	
4	Germany	0	0	1	1	
5	Japan	1	0	0	1	
6	Kazakhstan	0	0	1	1	
7	Russia	3	1	1	5	
8	South Korea	0	1	0	1	
9	United States	1	0	1	2	
10	Medals Totals For All Countries	5	5	5	15	