Hints for Credit Card Project, Fall 2025

Tong Yi

Contents

1 Hints for Task B: Read a CSV file and Calculate its Total

1.1 Related: Read a Tab-Separated-Values (TSV) file

The class `ifstream` is used to read plain text files, `ofstream` is used to write plain text files, and `fstream` is generally used for binary files such as audio or video. In this course, we will focus only on `ifstream` and `ofstream`.

When a computer reads a file, it behaves much like a human reader: it processes the file from top to bottom, and within each line, from left to right.

To read the contents of a TSV file (tab-separated values), we can use the extraction operator `>>` (also called the pull operator). This operator reads input until it encounters the next whitespace character, which could be a space, a tab, or a newline.

Step 1 Before reading a file, it is important to understand its structure:

How many non-data lines (such as headers or comments) appear before the actual data.

The order of the columns: for example, what the first column represents, what the second column represents, and so on.

The meaning of each column (e.g., date, description, category, amount).

The data type of each column (e.g., integer, floating-point, string).

Step 2 Open a plain text file to read by instantiating an ifstream object.

```
ifstream fin(fileName); //fileName is a string variable.
```

Step 3 Skip all explanation lines.

Step 4 Test the file can be read or not using fail method of fstream class. You may not be able to read the file since the file might be missing or corrupt, or lack of permission.

```
if (fin.fail()) {
   cerr << "The file cannot be opened." << endl;
   exit(1); //leave the program
}
```

Step 5 Use `fin >> variableName;` to read from the file associated with fin. Make sure the data value matches the type of `variableName`.

Step 6 After reading, use `fin.close();` to close the file associated with fin.

1.1.1 Example of Reading a Tab-Separated-Values (TSV) file

Suppose we have a TSV file that records each student's name, midterm score, final score, and bonus points. To calculate the total score, assign a weight of 30% to the midterm and 70% to the final, then add the bonus points.

1. Here are the contents of a file named `scores_f25.tsv`. The first line are column header. Starting from the second line, there are data. For example, the student's name is Ann. Her midterm grade is 77, her final grade is 78, and bonus is 1.2.

   Use text editor tools like Visual Studio Code (Mac and Windows) or TextEdit (Mac) or notepad++ (Windows) to create the following file, name it `scores_f25.tsv`. Put in the same folder of your C++ source code.

```
Name Midterm Final Bonus
Ann 77  78  1.2
Bob 88  89  2
Charles 99 100 0
```

2. First open the file using ifstream (input file stream). Need to import fstream library to use ifstream class.

   The name of ifstream object `fin` is to distinct from standard input stream object `cin`. You can rename `fin` with any meaningful name.

   Note that we do not need to instantiate `cin`, it always associated with keyboard input. The name `cin` cannot be changed.

   However, `ifstream` object `fin` needs to be instantiated as follows. And the name `fin` can be renamed as other meaningful names, for example, `inp`.

```
//instantiate an ifstream object fin that reads from scores_f25.tsv.
ifstream fin("scores_f25.tsv"); //ifstream means input file stream
```

The following statement instantiates a string object called `greeting`, whose initial value is `"hello, world"`.

```
string greeting("hello, world");
```

3. Test whether the file can be opened or not. Sometimes the file can be corrupted or you do not have authority to read it. If the file cannot be opened, we print out an error message, then quit the program by calling exit function.

```
if (fin.fail()) {
   //cerr is the destination of error messages.
   //By default, it is the console.
   cerr << "file cannot be open to read." << endl;

   //Stop running the program.
   //Exit to the operating system with error code 1.
   exit(1);
}
```

4. If the file can be opened properly, skip the first line "Name Midterm Final Bonus", since it does not contain data.

```
string line;
getline(fin, line);
//no further process of line is needed since line is column header,
//not the data we need in this program.
```

5. Every row of data starts with name (a string), midterm grade (an integer), and final grade (an integer), and a bonus (a double type number). Since each column is separated by a space, we can use >> operator.

   Here is the pesudocode of data processing.

```
as long as there is a name  //each data line starts with a name
begin
     read midterm
     read final
     read bonus

     calculate and print the total, which is 0.3 * midterm + 0.7 *
        final + bonus
end
```

   When writing the actual code, need to note that `final` is a keyword in C++ and cannot be used as a variable name. Here is an implementation.

   Note that we use `fin >> variableName` since the source of data comes from fin, not `cin`.

```
string name;
int midterm;
```

```
int finalGrade;
double bonus;
double total;
while (fin >> name) {
    fin >> midterm;
    fin >> finalGrade;
    fin >> bonus;

    total = 0.3 * midterm + 0.7 * finalGrade + bonus;

    cout << name << " " << total << endl;
}
```

6. Close the ifstream object fin after finishing reading the corresponding file.

```
fin.close();
```

7. Here is a **link** to work on a TSV file.

```
//code is in https://onlinegdb.com/Zf2Y52RQu

//contents of scores_f25.tsv

//Name midterm final bonus
//Ann 77  78  1
//Bob 88  89  2
//Charles 99 100 0.5

//Sample output:

//Ann 78.7
//Bob 90.7
//Charles 100.2

#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main() {
    //create an ifstream object fin that will read from scores_f25.tsv.
    ifstream fin("scores_f25.tsv");

    if (fin.fail()) {
       //cerr is the destination of error messages.
       //By default, it is the console.
       cerr << "file cannot be open to read." << endl;
```

```
29
30          //Stop running the program.
31          //Exit to the operating system with error code 1.
32          exit(1);
33      }
34
35
36      string line;
37      getline(fin, line);
38      //no further process of line is needed since line is column header,
39      //not the data we need in this program.
40
41      string name;
42      int midterm;
43      int finalGrade;
44      double bonus;
45      double total;
46      while (fin >> name) {
47          fin >> midterm;
48          fin >> finalGrade;
49          fin >> bonus;
50
51          total = 0.3 * midterm + 0.7 * finalGrade + bonus;
52
53          cout << name << " " << total << endl;
54      }
55
56      fin.close();
57
58      return 0;
59  }
```

1.2 Read a CSV file

The steps to read a CSV file is similar to those of reading a TSV file, they only differ in Step 5.

Step 5.1 For each row of data in a csv file, every item **except** the last one is ended by ','. Use `getline(fin, variableName, ',');` to read a value before ',' and put that value in `variableName`, a string type variable.

Step 5.2 The last item in a row of data is ended by a new line character. Use `getline(fin, variableName);` to read the value ofthe last item to appropriate `variableName`, a string variable.

Step 5.3 Unlike extraction operator `>>`, which can convert the read value into a corresponding type not limited to string, function `getline` reads a value to a string variable. To convert a string to an int, use function `stoi`, which takes in a string and converts to an integer. For example, `stoi("15")` returns 15.

Similarly, function `stod` converts a string to a corresponding double number. For example, `stod("123.56")` returns a double number 123.56.

1.2.1 Example of Reading a Comma-Separated-Values (CSV) file

Suppose we have a CSV file recording name, midterm, final and bonus of students. Calculate the total where the weight of midterm 30% and the of final is 70%, then plus bonus.

1. Here are the contents of a file named `scores_f25.csv`. The first line are column header. Starting from the second line, there are data. For example, the midterm grade of Ann is 77, her final grade is 78, and bonus is 1.2.

   Use text editor tools like Visual Studio Code (Mac and Windows) or TextEdit (Mac) or notepad++ (Windows) to create the following file, name it `scores_f25.csv`. Put in the same folder of your C++ source code.

   Read the contents of the following CSV file, a name may contain spaces, since the delimiter (separator) of column data is a comma, not a space.

```
Name,Midterm,Final,Bonus
Ann Johnson,77,78,1.2
Bob Smith,88,89,2
Charles Chan,99,100,0.5
```

2. First open the file using ifstream (input file stream). Need to import fstream library to use ifstream class.

   The name of ifstream object `fin` is to distinct from standard input stream object `cin`. You can rename `fin` with any meaningful name.

```
//create an ifstream object fin that will read from scores_f25.csv.
//if you test the code in onlinegdb, rename the file as scores_f25.txt.
ifstream fin("scores_f25.csv");
```

3. Test whether the file can be opened or not. Sometimes the file can be corrupted or you do not have authority to read it. If the file cannot be opened, we print out an error message, then quit the program by calling exit function.

```
if (fin.fail()) {
   //cerr is the destination of error messages.
   //By default, it is the console.
   cerr << "file cannot be open to read." << endl;

   //Stop running the program.
   //Exit to the operating system with error code 1.
   exit(1);
}
```

4. If the file can be opened properly, skip the first line "Name,Midterm,Final,Bonus", since it does not contain data.

```
string line;
getline(fin, line);
//no further process of line is needed since line is column header,
//not the data we need in this program.
```

5. Every row of data starts with name (a string), midterm grade (an integer), and final grade (an integer), and a bonus (a double type number). Since each column is separated by a space, we can use >> operator.

   Here is the pesudocode of data processing.

```
as long as there is a name  //the line has data, starting with column
   name
begin
     read a string and convert it to midterm
     read the next string and convert it to final
     read the next string and convert it to bonus

     calculate and print the total, which is 0.3 * midterm + 0.7 *
        final + bonus
end
```

   When writing the actual code, need to note that `final` is a keyword in C++ and cannot be used as a variable name. Here is an implementation.

   Note that we use `getline(fin, variableName, ',');` since the source of data comes from fin, and stop at the next ','.

```
string name;
string midtermStr;
int midterm;
string finalGradeStr;
int finalGrade;
string bonusStr;
double bonus;
double total;

//cannot use fin >> name; to replace getline(fin, name, ',')
//extraction operator >> stops before a white space symbol,
//which can be a space, a tab, or a new line character
while (getline(fin, name, ',')) {
    //cannot replace getline(fin, midtermStr, ','); with
    //fin >> midtermStr;
    //extraction operator >> reads all the contents until a white space
    getline(fin, midtermStr, ',');
    midterm = stoi(midtermStr); //midterm is an int

    getline(fin, finalGradeStr, ',');
```

```
    finalGrade = stoi(finalGradeStr);

    //getline(fin, bonusStr, ','); //WRONG, bonus is the last item in
       the row,
    //the delimiter after bonus is a new line character, not a ,
    getline(fin, bonusStr); //this version of getline reads bonusStr
       before the first new line character
    //use stod function to convert a string to a double number
    bonus = stod(bonusStr);

    total = 0.3 * midterm + 0.7 * finalGrade + bonus;

    cout << name << " " << total << endl;
}
```

6. Close the ifstream object fin after finishing reading the corresponding file.

```
fin.close();
```

7. Here is a **link** to work on a CSV file.

```
//onlinegdb link: https://onlinegdb.com/FNsFt0Hwf
//need to rename scores_f25.csv as scores_f25.txt in onlinegdb

//contents of scores_f25.csv

//Name,Midterm,Final,Bonus
//Ann Johnson,77,78,1.2
//Bob Smith,88,89,2
//Charles Chan,99,100,0.5

//Sample output:

//Ann Johnson 78.9
//Bob Smith 90.7
//Charles Chan 100.2

#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main() {
    //create an ifstream object fin that will read from scores_f25.csv.
    //if you test the code in onlinegdb, rename the file as scores_f25.
       txt.
    ifstream fin("scores_f25.csv");
```

```
26
27      if (fin.fail()) {
28          //cerr is the destination of error messages.
29          //By default, it is the console.
30          cerr << "file cannot be open to read." << endl;
31
32          //Stop running the program.
33          //Exit to the operating system with error code 1.
34          exit(1);
35      }
36
37
38      string line;
39      getline(fin, line);
40      //no further process of line is needed since line is column header,
41      //not the data we need in this program.
42
43
44      string name;
45      string midtermStr;
46      int midterm;
47      string finalGradeStr;
48      int finalGrade;
49      string bonusStr;
50      double bonus;
51      double total;
52
53      //cannot use fin >> name; to replace getline(fin, name, ',')
54      //extraction operator >> stops before a white space symbol,
55      //which can be a space, a tab, or a new line character
56      while (getline(fin, name, ',')) {
57          //cannot replace getline(fin, midtermStr, ','); with
58          //fin >> midtermStr;
59          //extraction operator >> reads all the contents until a white
                space
60          getline(fin, midtermStr, ',');
61          midterm = stoi(midtermStr); //midterm is an int
62
63          getline(fin, finalGradeStr, ',');
64          finalGrade = stoi(finalGradeStr);
65
66          //getline(fin, bonusStr, ','); //WRONG, bonus is the last item
                in the row,
67          //the delimiter after bonus is a new line character, not a ,
68          getline(fin, bonusStr); //this version of getline reads
                bonusStr before the first new line character
```

```
        bonus = stod(bonusStr); //bonus is a double

        total = 0.3 * midterm + 0.7 * finalGrade + bonus;

        cout << name << " " << total << endl;
    }

    fin.close();

    return 0;
}
```

2 Task C: Read a CSV file and Sum Up Entries in a Time Range

Here is a pseudocode of Task C.

```
#include <iostream> //std::cout, std::fixed, std::scientific
#include <string>
#include <fstream> //ifstream
#include <iomanip> //std::setprecision
using namespace std;

//convert to standard mm/dd/yyyy format
string convert(string original);

int main() {
    //TODO: enter file name

    //TODO: check whether the file can be opened or not.
    //      If not, print an error message and exit.

    //TODO: enter start and end date to specify time range

    //TODO: read the file and filter all the records that
    //      fits in the time range specified by start and end date.
    //      Calculate the total amount.
    //Warning: need to convert dates to standardized mm/dd/yyyy format.

    //TODO: close the file after finishing reading.

    //TODO: print out the total amount.


    return 0;
}

```

```
//convert from m/d/yy to standard mm/dd/yyyy format
string convert(string original) {
    //Note that string parameter original is declared in parameter list.
    //It is initialized when function convert is called.

    string result;

    //TODO: use substr method of string class
    //      to extract month, day, and year information.
    //      Use + (concatenate) operator to pad modified info to result.

    //TODO: what to do when month has only one digit?


    //TODO: what to do when day has only one digit?


    //TODO: when year has only two digits,
    //      if year is less than or equal to 25,
    //         concatenate "20" before the corresponding year string.
    //      otherwise, concatenate "19" before the corresponding year
       string.


    //TODO: concatenate processed month-, day-, and year- info
    //      to return a string in mm/dd/yyyy format.

    return result;
}
```

3 Hints for Task E: Monthly Category Total

3.1 Read a file for More Than Once

In this program, we need to read the file twice:

The first time is to find out categories in the file and save them in alphabetic order in an array, whose elements are then displayed for users to choose a category.

The second time is find out the monthly category total for the chosen category.

To read a file for more than once, there are two approaches: reopening the file, or rewinding the pointer of the file to the beginning after it reaches the end.

Here's a breakdown of both approaches:

1. Reopening the File

   This approach involves opening the file, reading its contents, closing it, and then repeating the process when you need to access the file again. Steps:

(a) Open the file using an ifstream object.

(b) Check if the file opened successfully.

(c) Read the contents of the file until the end.

(d) Close the file.

(e) Repeat steps i-iv for subsequent readings.

2. Instead of closing and reopening the file, you can reposition the file pointer to the beginning of the file using the seekg() and clear() methods of the ifstream object. Steps:

   (a) Open the file using an ifstream object. Suppose the object is called `fin`.

   (b) Check if the file opened successfully.

   (c) Read the contents of the file until the end.

   (d) Reset the file stream's error flags using `fin.clear()`. Reposition the file pointer to the beginning using `fin.seekg(0)`.

   (e) Repeat steps iii-iv for subsequent readings.

   (f) Call close method for `fin`.

3.2 Create Array to Hold Categories

Here are main steps.

1. Use insertion sort to create a string array to hold unique categories in dictionary (alphabetic) order.

2. Assume that there are no more 20 categories in a credit card file. As a result, it is sufficient to create an array of string with capacity 20 to save the categories in that file.

3. In the very beginning, there is no element in the array yet, initialize its size to be zero.

4. Insert one element a time, until there is no more element to insert or the capacity of the array is reached.

5. In each insertion, make sure that no redundant element is inserted and the array elements are maintained in dictionary order.

We give a similar example to read a file and save scores in non-descending order of array. Unlike our project, redundant elements are allowed in this example.

Explain key parts of insertion sort.

1. If the array is full, print an error message and exit to the operating system with an error code 1.

```
50          if (size == NUM_SCORES) {
51             cerr << "scoreArr is full and cannot hold any more element"
                  << endl;
52             exit(1);
53          }
```

2. When array `scoreArr` has not reached its capacity, insert the current element `score` to the array, which is sorted in non-descending order already. After insertion, the array remains to be sorted.

   (a) Find the Insertion Index:

      i. Iterate through scoreArr from the beginning.

      ii. Stop at the first element that is greater than or equal to score. This is the ideal insertion point.

      iii. If no such element is found (i.e., score is greater than all existing elements), the new score should be appended to the end.

```
65          int i = 0;
66          while (i < size && scoreArr[i] < score)
67              i++;
```

   Note that
   `(i < size && scoreArr[i] < score)` cannot be rewritten as
   `(scoreArr[i] < score && i < size)`. Whenever we use an array index, say `i` in `scoreArr[i]`, need to make sure the index is valid. That is, `0 <= i` and `i < size`. Since `i` starts from 0 based on `int i = 0;` and keeps on increasing in each loop as shown in `i++;`, just need to check `i < size`.

3. Once the above loop finishes, either `i == size` or `i < size and scoreArr[i] >= score`.

   (a) When `i == size`, either the array has no element yet or all elements in `scoreArr` are smaller than `score`, just insert `score` to the end of the array, the corresponding index is `size`.

   (b) Otherwise, `i < size and scoreArr[i] >= score`, that is, `scoreArr[i]` is the first element that is larger than or equal to `score`. So `score` needs to be inserted in index `i`. So all elements indexed at `i` and above need to be moved to the right one spot.

   (c) Move the element indexed at `j` right one spot is achieved by `rr[j+1]␣=␣rr[j];*`. The indices of the elements to be moved range from `i` to `size-1`, the last index.

   (d) The order of moving is important. Start moving from the element in the end. Imagine a student to be insert into the middle of a queue, first move the student at the end to right, then move the student second-to-the-end to right, $\cdots$, the last element to move to the right is the one indexed at `i`.

```
77          if (i == size) { //insert score to the end of the array
78             scoreArr[size] = score;
79          }
80          else {
81              for (int j = size-1; j >= i; j--)
82                  scoreArr[j+1] = scoreArr[j];
83
84              scoreArr[i] = score;
85          }
```

   Here is complete code to extract numbers from the `Score` column and store them in an array, sorted in ascending order. See **link**. In onlinegdb, file with csv suffix is not recognized, so we change the suffix of the text file to txt.

Test with a csv file with following contents,

```
Name,Score
Ann Johnson,92
Bob Smith,78
Charles Chan,88
Deb Ahmed,78
```

```
//https://onlinegdb.com/h3iHNeZYU
//Purpose: read a file and save scores in non-descending order of
   array
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

//contents of scores.csv

//Name,Score
//Ann Johnson,92
//Bob Smith,78
//Charles Chan,88
//Deb Ahmed,78

int main() {
    cout << "enter a csv file: ";
    string fileName;
    cin >> fileName; //cin << fileName; has error

    ifstream fin(fileName);

    if (fin.fail()) {
       cerr << fileName << " cannot be opened." << endl;
       exit(1);
    }

    //skip the first line, which is column header
    string line;
    getline(fin, line);

    string name;
    string scoreStr;
    int score;

    //Assume that there are at most 100 scores in a file
    const int NUM_SCORES = 100;
    int scoreArr[NUM_SCORES];
```

```
39      int size = 0; //current number of scoreArr,
40          //or the index of the next element to write to scoreArr
41
42      //read from fin, stop at the first ',',
43      //save the data in name
44      while (getline(fin, name, ',')) {
45          getline(fin, scoreStr); //scoreStr stops at '\n'
46
47          //convert string scoreStr to an int
48          score = stoi(scoreStr);
49
50          //scoreArr cannot hold more elements
51          if (size == NUM_SCORES) {
52             cerr << "scoreArr is full and cannot hold any more
                   element" << endl;
53             exit(1);
54          }
55
56          //assume that scoreArr is sorted,
57          //insert score to it so scoreArr is still sorted
58
59          //find out the first index i that scoreArr[i] >= score.
60          //That is, every element in subarray scoreArr[0..i-1],
61          //ie, every element scoreArr[0], scoreArr[1], ..., scoreArr
                [i-1],
62          //is < score,
63          //and scores[i] >= score
64
65          int i = 0;
66          while (i < size && scoreArr[i] < score)
67              i++;
68
69          //when the above loop stops,
70          //either i == size or
71          //i < size and scoreArr[i] >= score
72          //push all elements from index i all the way to the end one
                spot to the right
73          //That is, from the last element -- index size-1 --
74          //down to the ith element,
75          //move one spot right.
76
77          if (i == size) { //insert score to the end of the array
78             scoreArr[size] = score;
79          }
80          else {
81              for (int j = size-1; j >= i; j--)
```

```
                scoreArr[j+1] = scoreArr[j];

            scoreArr[i] = score;
        }
        size++;
    }

    fin.close();

    //display elements in scoreArr
    for (int i = 0; i < size; i++)
        cout << scoreArr[i] << endl;

    return 0;
}
```

3.3 Calculate and Print Monthly Category Total

Here are main steps.

1. Assume that there are at most 12 different months in a given file. Create an array of double type to save the monthly category sum with capacity 12. The first element of this array is the total for the input category in January, the second element is the total of the input category in February, $\cdots$, and the last element is the total of the input category in December.

2. Set the elements of the array properly.

3. Display the elements of this array.

   To pass gradescope, the format should be the month name (for example, Jan or January) followed by spaces, then monthly category total, displayed with two decimal places.

4. To left align a label or value in C++:

   (a) Include the `<iomanip>` (Input/Output Manipulator) header.

   (b) Use `std::cout << std::left << std::setw(width) << variable;`.

      i. `std::left` sets the alignment to the left.

      ii. `std::setw(width)` sets the minimum field width, a positive integer. The output will be padded with spaces (by default) to fill the width if the content is shorter.

      iii. variable is the item you want to print.

5. To right align, follow the same steps but use std::right instead.