CS201 – Spring 2015-2016 - Sabancı University

Homework #7 – Board Game Sales

Due May 15, 2015, Sunday, 22:00 (Sharp Deadline)

Brief Description

In this homework, you will write a program that produces an output based on data read from an input data file. Data file will include the monthly sales information for one year, about several board games from various companies. Your program will read monthly sales count for each board game, by processing these numbers, annual total sales counts will be calculated, and your program will produce a sales report which is ordered by annual sales counts (descending order). The details of the input and output file formats, and the rules of processing will be explained below.

Name and Structure of Database (Input) File

The data is stored in a single text file. It is generated outside of the program using Notepad (we provide example input files within the zip package of this homework).

At the beginning of your program, user will be asked to enter an input file name for the sales file. If the user enters a name for a file that does not exist, your program should display an appropriate message and ask for the name of a new input file name until the file is successfully opened.

The sales input file contains sales information for each board game. For a single record, there are four columns of information on the same line, separated by a semicolon followed by white space(s). Please note that, for each board game there may be multiple records corresponding to different months. The record structure is described as follows:

Boardgame_Name;	*Month_Number;*	*Company_Name;*	*Sales_Count*

- Record line starts with the name of the board game. It may consist of several words.
- Second data item of the record is month number, which is between 1 and 12. You may assume that this information is in the correct format, and you do not need to make an input check for it.
- Third data item is company name. It may consist of one or multiple words.
- Last data item is the sales count (integer) for that board game in the given month.

The abovementioned record structure, which contains 4 different data items in one line, is just monthly information for a single board game. The input file contains information about several board games as well as multiple records for the same board game. Therefore, there are several such records in the input file. You may assume there is a record in each line of input file, no line is empty.

There might be any number of white spaces (i.e. blanks) between and after each data item in the input file. You should handle it. Remember that the data items are separated by semicolon.

The matching of board game names should be case-insensitive. That is, a board game name such as “MoNoPoLy” in one line and “MONOPOLY" in another line **SHOULD be considered the same**. Company names should also be case-insensitive. You should print board game names and company names in all upper-case letters to the output file. Therefore, you may prefer to convert them to all upper-case in the beginning. This may help you to match names as well.

There is no specific order of records in the input file. That means you cannot assume that the records of the input files are ordered according to *Boardgame_Name*, *Company_Name* or any other data attribute.

The structure of the input files and the associated rules and assumptions explained above are fixed: you cannot change them or make any other assumptions. No format check for the contents of the input files is needed: you may assume that all data are in the correct format. You may examine the sample input file (boardgames.txt) provided in the zip package for this homework (hw7.zip).

Processing

You **MUST** use a `struct` data structure to encapsulate data of one board game; such as the board game’s name, company’s name, its sales count etc. It is up to you to decide which fields are to be included in this `struct`.

You **MUST** use a `vector` to store several board games’ data. Of course, the element type of this vector will be the previously defined board game `struct`. Here please remark that you CANNOT make any assumptions about the number of board games. Therefore, your program should work for any number of board games. Moreover, your program should work for any board game names. In the sample, we used real board game names, but we can use other names for grading purposes.

Flow of the program may be as follows:

- Your program should first ask for file name information from user for input file. For the given input file name, you should try to open the file; if it fails your program should ask for another name until a valid file name is entered.
- After that, your program will ask for an output file name. Details regarding to this output file is explained in the following Output section.
- Then, your program processes the input file. The processing method could be word by word or line by line.
    - As mentioned above, there are several records for a board game in the input file. The processing for each board game is as follows:
        - If this board game is not in the vector (i.e. if this is the first appearance of the board game), then create a new board game `struct` for this board game and add to the vector.
        - If this board game is already in your vector, update this existing `struct`. That is, if this board game has occurred before in the yearly sales count calculations, then there should be a `struct` in the vector for it. If this is the case, you should only update the yearly sales count information about the board game without adding a `struct` for this board game to the vector again.

- After your program finishes reading and processing all of the sales information from the input file and storing the necessary data in the vector, it should sort the vector in **descending** manner.
    - You may modify and use one of sorting algorithms discussed in the lecture and/or provided by the book. If you want to develop your own sorting algorithm, of course you may, but this will be more difficult.
    - You should sort the vector by yearly sales count; board games with higher number of sales must be at higher positions in the table. If multiple board games have the same sales count, then they should be sorted by the company name; alphabetically smaller name (in string comparison terms) must be at a higher position in the table. If two or more of the board games with equal number of yearly sales belong to the same company, then they should be sorted by the board game name. Again, alphabetically smaller name (in string comparison terms) must be at a higher position in the table.
- Finally display a message that informs the user about the end of the process and the name of the output file.

Output

Output file name is asked from the user in the beginning of the program right after the input name is taken and input file is successfully opened.

After the board games are sorted, your program should print the sorted ranking list into a file. The name of the output file should also be given as an input by the user.

The format of the table is described below and **you must follow this format precisely**.

The first line of the output file is the header line of the table. This line is fixed and given below.

```
RANK, COMPANY, NAME, SALES
```

After this header line, the sales counts table is displayed in rank order (from top to bottom). Rank values start from 1. You should display each board game data in a separate line.

For each board game, 4 pieces of information must be displayed in the output file; these are:

Rank of the board game, Company Name, Board Game *Name, Yearly number of sales*

These 4 pieces of information must be displayed in this order and there must be commas <u>between them</u>. Please note that there's also one space after the commas.

The board game names and company names **MUST** be output in all **UPPER-CASE** no matter how it is spelled in the input file. For example, the board game name "MoNoPoLy" in one line in the input file and "MONOPOLY" in another line in the input file are the same and it should be printed as "MONOPOLY" in the output file.

Sample Input and Output Files

In the zip package of this homework, we provide sample input and output files: `boardgames.txt, output.txt`. You may examine them to understand the homework and output format in detail.

Sample Runs

The *italic* and **bold** phrases are inputs taken from the user.

Here is an example run of the program to show file opening error handling cases:

```
Please enter a filename for Board Game file: board.txt
Can not find the specified file.
Please enter a filename for Board Game file: dgame.txt
Can not find the specified file.
Please enter a filename for Board Game file: boardgames.txt
Please enter a filename for output file: output.txt
Results are in the file: output.txt
```

VERY IMPORTANT!

Your programs will be compiled, executed and evaluated automatically; therefore you should definitely follow the rules for prompts, inputs and outputs. See

- **Order of inputs and outputs** must be in the abovementioned format.
- **Prompts before inputs and outputs** must be **exactly the same** with examples.

Following these rules is crucial for grading, otherwise our software will not be able to process your outputs and you will lose some grades in the best scenario.

IMPORTANT!

If your code does not compile, you will get **zero**. Please be careful about this and double check your code before submission.

General Rules and Guidelines about Homeworks

The following rules and guidelines will be applicable to all homeworks, unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) of CS201. Office hours of TAs are at the class website. Recitations will partially be dedicated to clarify the issues related to homework, so it is to your benefit to attend recitations.

What and Where to Submit

Please see the detailed instructions below/in the next page. The submission steps will get natural/easy for later homeworks.

Grading and Objections

Careful about the semi-automatic grading: Your programs will be graded using a semi-automated system. Therefore you should follow the guidelines about input and output order; moreover you should also use same prompts as given in the Sample Runs. Otherwise semi-automated grading process will fail for your homework, and you may get a zero, or in the best scenario you will lose points.

Grading:

- ❑ Late penalty is 10% off of the full grade and only one late day is allowed.
- ❑ **Having a correct program is necessary, but not sufficient to get the full grade. Comments, indentation, meaningful and understandable identifier names, informative introduction and prompts, and especially proper use of required functions, unnecessarily long program (which is bad) and unnecessary code duplications (which is also bad) will also affect your grade.**
- ❑ Please submit your own work only (even if it is not working). It is really easy to find out "similar" programs!
- ❑ For detailed rules and course policy on plagiarism, please check out http://myweb.sabanciuniv.edu/gulsend/su_current_courses/cs-201-spring-2008/plagiarism/ and keep in mind that…

Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.

Grade objections: It is your right to object to your grade if you think there is a problem, but before making an objection please try the steps below and if you still think there is a problem, contact the TA that graded your homework from the email address provided in the comment section of your announced homework grade or attend the specified objection hour in your grade announcement.

- Check the comment section in the homework tab to see the problem with your homework.
- Download the .zip file you submitted to SUCourse and try to compile it.
- Check the test cases in the announcement and try them with your code.
- Compare your results with the given results in the announcement.

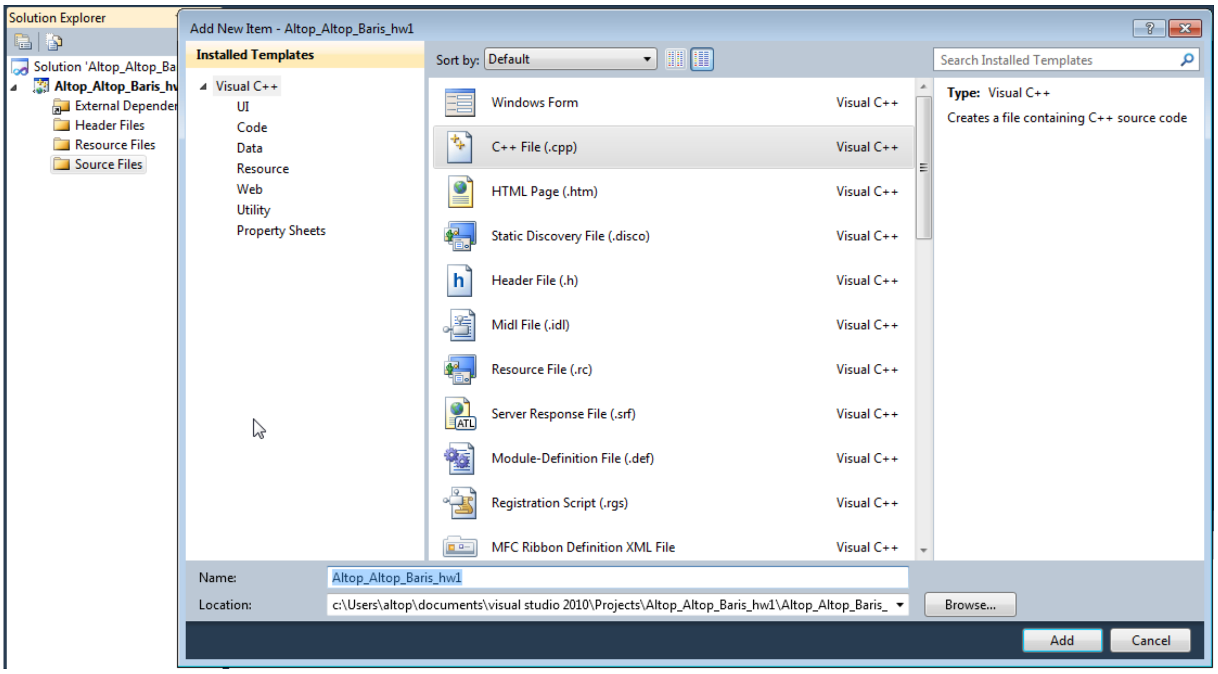
What and where to submit (IMPORTANT)

Submissions guidelines are below. Most parts of the grading process are automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Add your name to the program: It is a good practice to write your name and last name somewhere in the beginning program (as a comment line of course).

Name your submission file:

- ❑ Use only English alphabet letters, digits or underscore in the file names. Do not use blank, Turkish characters or any other special symbols or characters.
- ❑ Name your cpp file that contains your program as follows.
  "**SUCourseUserName_YourLastname_YourName_HWnumber.cpp**"



Windows Form	Visual C++
C++ File (.cpp)	Visual C++
HTML Page (.htm)	Visual C++
Static Discovery File (.disco)	Visual C++
Header File (.h)	Visual C++
Midl File (.idl)	Visual C++
Resource File (.rc)	Visual C++
Server Response File (.srf)	Visual C++
Module-Definition File (.def)	Visual C++
Registration Script (.rgs)	Visual C++
MFC Ribbon Definition XML File	Visual C++

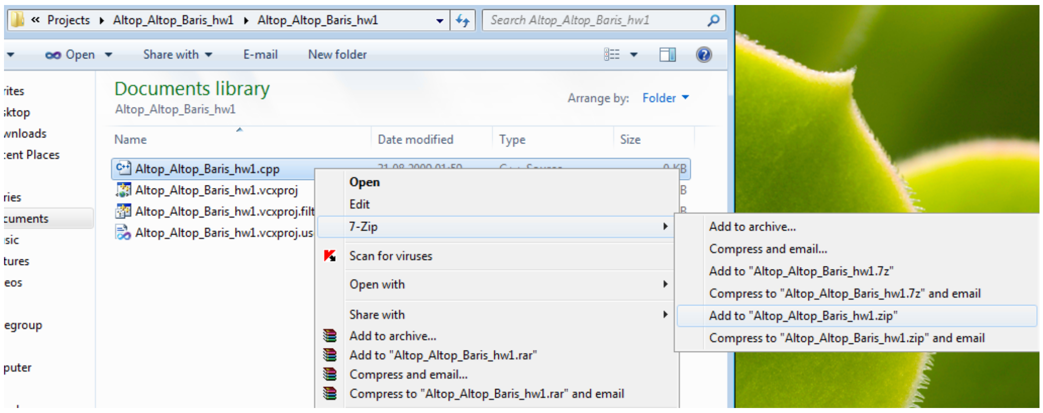
Type: Visual C++
Creates a file containing C++ source code

Name: Altop_Altop_Baris_hw1
Location: c:\Users\altop\documents\visual studio 2010\Projects\Altop_Altop_Baris_hw1\Altop_Altop_Baris_

- ❑ Your SUCourse user name is actually your SUNet user name which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

  Cago_Ozbugsizkodyazaroglu_Caglayan_hw7.cpp

- ❑ Do not add any other character or phrase to the file name.
- ❑ Make sure that this file is the latest version of your homework program.
- ❑ Compress this cpp file using WINZIP or WINRAR programs. Please use "**zip**" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only with zip files. Therefore, make sure that the resulting compressed file has a zip extension.


Projects ▸ Altop_Altop_Baris_hw1 ▸ Altop_Altop_Baris_hw1
Search Altop_Altop_Baris_hw1
Open
Share with
E-mail
New folder
Documents library
Altop_Altop_Baris_hw1
Arrange by: Folder
Name
Date modified
Type
Size
Altop_Altop_Baris_hw1.cpp
Altop_Altop_Baris_hw1.vcxproj
Altop_Altop_Baris_hw1.vcxproj.filt
Altop_Altop_Baris_hw1.vcxproj.us
Open
Edit
7-Zip
Scan for viruses
Open with
Share with
Add to archive...
Add to "Altop_Altop_Baris_hw1.rar"
Compress and email...
Compress to "Altop_Altop_Baris_hw1.rar" and email
Add to archive...
Compress and email...
Add to "Altop_Altop_Baris_hw1.7z"
Compress to "Altop_Altop_Baris_hw1.7z" and email
Add to "Altop_Altop_Baris_hw1.zip"
Compress to "Altop_Altop_Baris_hw1.zip" and email

❑ Check that your compressed file opens up correctly and it contains your **cpp** file. You will receive no credits if your compressed zip file does not expand or it does not contain the correct file.

❑ The naming convention of the zip file is the same as the cpp file (except the extension of the file of course). The name of the zip file should be as follows.

“**SUCourseUserName_YourLastname_YourName_HWnumber.zip**”

For example zubzipler_Zipleroglu_Zubeyir_hw7.zip is a valid name, but hw7_hoz_HasanOz.zip, HasanOzHoz.zip are NOT valid names.

<u>Submission:</u>

❑ Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

1) Click on "Assignments" at CS201 SUCourse (not the CS201 web site).
2) Click Homework 7 in the assignments list.
3) Click on "Add Attachments" button.
4) Click on "Browse" button and select the zip file that you generated.
5) Now, you have to see your zip file in the "Items to attach" list.
6) Click on "Continue" button.
7) Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

<u>Resubmission:</u>

❑ After submission, you will be able to take your homework back and resubmit. In order to resubmit, follow the following steps.

1) Click on "Assignments" at CS201 SUCourse.
2) Click Homework 7 in the assignments list.
3) Click on "Re-submit" button.
4) Click on "Add/remove Attachments" button
5) Remove the existing zip file by clicking on "remove" link. This step is very important. If you do not delete the old zip file, we receive both files and the old one may be graded.
6) Click on "Browse" button and select the new zip file that you want to resubmit.
7) Now, you have to see your new zip file in the "Items to attach" list.
8) Click on "Continue" button.
9) Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Ece Egemen and Gülşen Demiröz