CS201 – Fall 2019 - Sabancı University

Homework #5: Student Grade Ranking

Due December 4, Wednesday, 23:00

Brief Description

In this homework, you will write a program to generate a sorted student points table after reading grades of students from a file. This points table consists of information about the students ranking for a specific course. Your program will read students with id information from an input file and their grades from the second input file. By processing these course results, for each student, your program will compute total points by computing the weighted average of assignments, midterms and a final of the course. Then you will generate a sorted points table and when the user asks for a specific rank, your program displays the information (ID, name and total points) of that student at that rank. The details of the input and output file formats, and the rules of processing will be explained in this document.

We will be automatically grading your homework using GradeChecker, so it is very important to satisfy the exact same output given in the sample runs. You can utilize GradeChecker (http://sky.sabanciuniv.edu:8080/GradeChecker/) to check whether your code is working in the expected way. To be able to use GradeChecker, you should upload all of your files used in the homework. Additionally, you should submit **ALL** of your files (prompt.cpp, prompt.h, strutils.cpp and strutils.h) to SUCourse **without zipping** them. **Just a reminder, you will see a character ¶ which refers to a newline in your expected output.**

The name of your main source (cpp) file should be in the expected format: "sucourseusername_lastname_name_hwnumber.cpp" (all lowercase letters). Please check the submission procedures of the homework in this document.

To get help using GradeChecker you may ask questions to the list of your grader TAs: cs201gchelp@lists.sabanciuniv.edu

NOTE THAT, we did not upload all of the queries in the sample runs to GradeChecker. So, you may want to check all of the sample runs manually.

IMPORTANT: In order to use GradeChecker for this homework, you need to upload everything including students.txt, grades.txt, prompt.cpp, prompt.h, strutils.cpp and strutils.h (if used). You MUST submit ALL of your files (such as *prompt.cpp, prompt.h, strutils.cpp, strutils.h*) **to SUCourse without zipping them.** We should be able to compile/build your files.

Input File

Your program should read two input files; first one includes the names (i.e. first and last name) of the students with their ID numbers and the other one contains the grades of the corresponding students with their ID information (i.e. only the ID number is unique not the first or last names of the students).

In the student information file, there are white spaces in the student names. There is a single space between the first name, middle name and last name of the student to separate these. If a student does not have a middle name, then there will be only one single space between the name and the last name of the student (i.e. a student should have a first name as well as a last name but may not have a middle name). However, we do not use non-English letters, like ş, ö, ü, ğ, ı, ç, etc. in the student names. In this homework, you may assume that student names are given in this way (i.e. with blank information as previously explained and non-English letters are not used).

An example students information file is shown below:

```
00009029 Ali Can Akdeniz
00012287 Onur Akdeniz
00011412 Yonca Betul Karadeniz
00010206 Gizem Gezici
```

In the grades file, there are grade results of different evaluation parts of a specific course (e.g. homework, midterm and final). In order to differentiate these, each evaluation scheme results begins with a constant line of one of these below:

*****HOMEWORK*****
*****MIDTERM*****
*****FINAL*****

After this line, there comes the id number of a student and the grade of the corresponding student for a particular assessment of the course line by line. There is only one student (i.e. id number of that student) on each line and his/her grade information for that particular assessment of the course. At the end of the id-grade information of the students for a specific assessment, there is an empty line.

Part of an example grade results file is shown below:

```
***HOMEWORK***
00011530 82
00009962 86

***MIDTERM***
00011530 55
00010206 75
00009568 90

***MIDTERM***
00009568 85
00010206 60

***FINAL***
00009962 70
00011366 50
00011530 60
```

The number of students in the grades list may vary in each particular assessment of the course. Moreover, there may be several assignments (i.e. homework) or midterms of a specific course. Therefore, you cannot make any assumption about the number of specific evaluation schemes as well as the number of students in each evaluation criteria. Furthermore, you cannot make any assumption about the order of these assessments.

Obviously, a particular student is listed at most one time in a particular assessment, but a student may be absent in some of these assessments (i.e. student may not take the midterm1 assuming that no make-up or s/he may not submit one of the assignments). A student may be completely absent in the grades file, in this case he/she should not appear in the grades points table at all.

For the sample input files which contains the students and grades information, you may check out the students.txt and grades.txt files provided in this homework google drive folder respectively. However, please remark that the students and the grades file which will be used for grading your assignments may be different than this sample. **THEREFORE, DO NOT MAKE ANY ASSUMPTIONS ABOUT THE CONTENT OF THESE FILES; JUST READ THE STUDENTS AND GRADES FILES AND PROCESS THEM.**

The structure of the input file, the associated rules, assumptions and the things that you cannot assume, which are all explained above, are fixed such that you cannot change them or make any other assumptions. No format check for the contents of the input file is needed. You may assume that all data are in correct format.

The names of both of the input files are *keyboard* inputs. If the user enters a wrong file name that does not exist, then your program should display an appropriate error message (check samples for error message) and terminate the program immediately. It is not needed to (and please do not) ask for a new input file name.

Processing

You can use a struct data structure to encapsulate data of one student, such as student name, points, etc. It is up to you to decide which fields are to be included in this `struct`.

You can use a vector to store several students' data. Of course, the element type of this vector will be the previously defined student `struct`. Here please remark that you CANNOT make any assumption about the number of students in the course. The sample grades data file that we provided in the google drive folder is partially taken from the results of a course. You can observe from this sample that there are different numbers of students in each assessment list. Therefore, your program should work for any number of students. Moreover, your program should work for any student names which are consistent with the naming rules discussed in the *Input File* Section. In the sample, we used real student names from a previous class, but we can use other names for grading purposes.

The points table is a sorted list of students rankings. The order is determined by the total points that each student gained through all course assessments in a specific course. If there is a tie in points meaning there are two students with the same points, some other criteria will be used to determine the order. These details will be explained below.

Flow of the program may be as follows:

- Your program should first ask for file name information from the user for the input files. Then, the user enters these file names from the keyboard. After that, your program opens the input files and tests for failure. If it fails to open any of these files, then the program will terminate with an appropriate message (see sample runs for the message).
- If it opens all of the files successfully, then your program processes the input files. The processing method will be line by line since there is data for a specific student on each line for the input files of *student information* and *grade results.*
    - Regarding processing the input files, first, you need to calculate the *total points that a student achieved* at the end of the course by using ID information since there is no name information in the *grade results* file (i.e. also name information is not unique to search for a student).
    - As mentioned in the *Input File* Section, there are grades for different assessments of a course in the input file of *grade results*. A particular assessment starts with ***Homework***, ***Midterm***, or ***Final*** tag. After one of these tags, we have the student ID number and his/her grade for the corresponding assessment (one student data per line). You need to differentiate these tags as you read them since each of them has a distinct weight and will contribute to the *total points* of a student at various levels. The *total points* of the student will be the *weighted sum of the grades* that s/he achieved for these specific evaluations (i.e. tags).
    - The processing for each student of the particular assessment section in the *grade results* file is as follows (note that this is just a suggested algorithm, and there could be other ways to solve this, so please feel free to create your own algorithm as well as using this one):
        - If this student is not in the vector (i.e. if this is his/her first appearance in the *grade results* file), then create a new student struct for this student and add to the vector. As you process the *grade results* file, you should retrieve the names of the corresponding student on that line from the *students* file by using the ID number (i.e. ID number exists in both of the input files).
        - If this student is already in your vector, update this existing student struct. That is, if this student has occurred before in the *grade results* file, then there should be a struct in the vector for him/her. If this is the case, you should only update the necessary information about the corresponding student without adding a struct for him/her to the vector again.
        - Updating the data of a specific student: Update the total point of a student by adding the *weighted sum of the points* gained from the particular assessment to his/her previous points. The weights for each evaluation scheme of a course are given below. The number of assignments (i.e. homework) and midterms will be more than one since the sum of the weight percentages should be equal to 100% (i.e. overall course grade will be computed from all of these particular assessments). You may assume the total percentage of these in the file will sum up to 100%, however do not make any assumptions of the number of each assessment. For example; there could be 3 homework assignments, 2 midterms and 1 final as given in the grades.txt file OR 1 homework, 3 midterms and 1 final OR 5 midterms and 0 homework and final as given in the grades2.txt file in the homework drive folder.

Particular Assessment	Weight Percentage
Homework	10%
Midterm	20%
Final	30%

- After your program finishes reading and processing all of the information in the input file of *grade results* 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 provided by the book and/or discussed in the lecture. 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 *points*; students with higher *points* must be at higher positions in the table. If multiple students have the same *point*, then you should use the *last name* of the student; in this case the student with alphabetically smaller last name (in string comparison terms) must be at a higher position in the table. If multiple students have both the same *point* and *last name*, then they should be in any order in the table. You do not need to worry about it.
- Finally you display the message "*Enter the rank you want to query (enter 0 to exit):*" that asks the user to enter the rank number to query for. You will display this query message until the user enters 0 to exit the program.

Output

After your program has the sorted point table and the user enters a rank integer, the student information with the given rank is displayed on the screen. Four pieces of information must be displayed for the student queried; these are *Rank of the student in the points table*, *ID number of the student, Name of the student (i.e. first name, middle name if exists and last name), Points that the student gained*. **These four pieces of information must be displayed in this order and there must be commas in between.**

For the name of the student in the output, please use the following formats for the possible two cases (NO ADDITIONAL SPACES!):

i. FirstName-SingleSpace-MiddleName-SingleSpace-LastName (middle name exists)
   Example: `Ali(space)Can(space)Akdeniz`

ii. FirstName-SingleSpace-LastName (there is no middle name)
   Example: ~~`Gizem(space)(space)Gezici`~~ (wrong), it should be `Gizem(space)Gezici` (right)

In the google drive folder of this homework, we provide sample input files (students.txt, grades.txt with an example ranking table file also (results.txt). You may examine and of course use them to understand the homework and output format in detail.

Please see the sample runs for examples using these given input files.

Sample Runs

Below, we provide a sample run of the program that you will develop. The italic and bold phrases are inputs taken from the user. You should follow the input order in these examples and the prompts your program will display must be **exactly the same** as in the following examples.

Sample Run 1

```
Please enter a filename for Students Grades Results: grades
Can not find the requested file. Terminating application ...
```

Sample Run 2

```
Please enter a filename for Students Grades Results: grades.txt
Please enter a filename for Students Names : nothere.txt
Can not find the requested file. Terminating application ...
```

Sample Run 3

```
Please enter a filename for Students Grades Results: grades.txt
Please enter a filename for Students Names : students.txt
Enter the rank you want to query (enter 0 to exit): 1
1, 00012287, Onur Akdeniz, 81.2
Enter the rank you want to query (enter 0 to exit): 4
4, 00011464, Alper Aydin, 74.7
Enter the rank you want to query (enter 0 to exit): 4
4, 00011464, Alper Aydin, 74.7
Enter the rank you want to query (enter 0 to exit): 19
Rank needs to be greater than 0 and smaller than 16!
Enter the rank you want to query (enter 0 to exit): 11
11, 00010926, Ekrem Sinan Aygun, 60.5
Enter the rank you want to query (enter 0 to exit): 7
7, 00011530, Burcu Aciksoz, 66.2
Enter the rank you want to query (enter 0 to exit): 8
8, 00010206, Gizem Gezici, 66.2
Enter the rank you want to query (enter 0 to exit): 14
14, 00011412, Yonca Betul Karadeniz, 57.5
Enter the rank you want to query (enter 0 to exit): 2
2, 00009029, Ali Can Akdeniz, 74.9
Enter the rank you want to query (enter 0 to exit): 15
15, 00008963, Selcuk Akaydin, 50.8
Enter the rank you want to query (enter 0 to exit): -5
Rank needs to be greater than 0 and smaller than 16!
Enter the rank you want to query (enter 0 to exit): 314323
Rank needs to be greater than 0 and smaller than 16!
Enter the rank you want to query (enter 0 to exit): 5
5, 00009713, Ozge Karadeniz, 68.9
Enter the rank you want to query (enter 0 to exit): 0
Exiting...
```

Sample Run 4

```
Please enter a filename for Students Grades Results: grades2.txt
Please enter a filename for Students Names : students.txt
Enter the rank you want to query (enter 0 to exit): 15
Rank needs to be greater than 0 and smaller than 14!
Enter the rank you want to query (enter 0 to exit): 13
13, 00011366, Yigit Akar, 54.6
Enter the rank you want to query (enter 0 to exit): 1
1, 00012287, Onur Akdeniz, 83
Enter the rank you want to query (enter 0 to exit): 5
5, 00009713, Ozge Karadeniz, 72.4
Enter the rank you want to query (enter 0 to exit): 0
Exiting...
```

Use of Functions and Other Rules

Unlike the second homework, we will not specify any functions here. But you are expected to use functions to avoid code duplication and improve the modularity of your program. **If your main function or any user-defined function is too long and if you do everything in main or in another user-defined function, your grade may be lowered.**

AND PLEASE DO NOT WRITE EVERYTHING IN MAIN AND THEN TRY TO SPLIT THE TASK INTO SOME FUNCTIONS JUST TO HAVE SOME FUNCTIONS OTHER THAN MAIN. THIS IS TOTALLY AGAINST THE IDEA OF FUNCTIONAL DESIGN AND NOTHING BUT A DIRTY TRICK TO GET SOME POINTS. INSTEAD PLEASE DESIGN YOUR PROGRAM BY CONSIDERING NECESSARY FUNCTIONS AT THE BEGINNING.

Try to use parametric functions and avoid inputs in functions. Do NOT use any global variables (variables defined outside the functions) to avoid parameter use.

As in the previous homework, no abrupt program termination is allowed in the middle of the program! The program flow should continue until the end of the main function.

No abrupt program termination please!

You may want to stop the execution of the program at a specific place in the program. Although there are ways of doing this in C++, it is not a good programming practice to abruptly stop the execution in the middle of the program. Therefore, your program flow should continue until the end of the main function and finish there.

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.

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:

- **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 all .cpp and .h files 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.

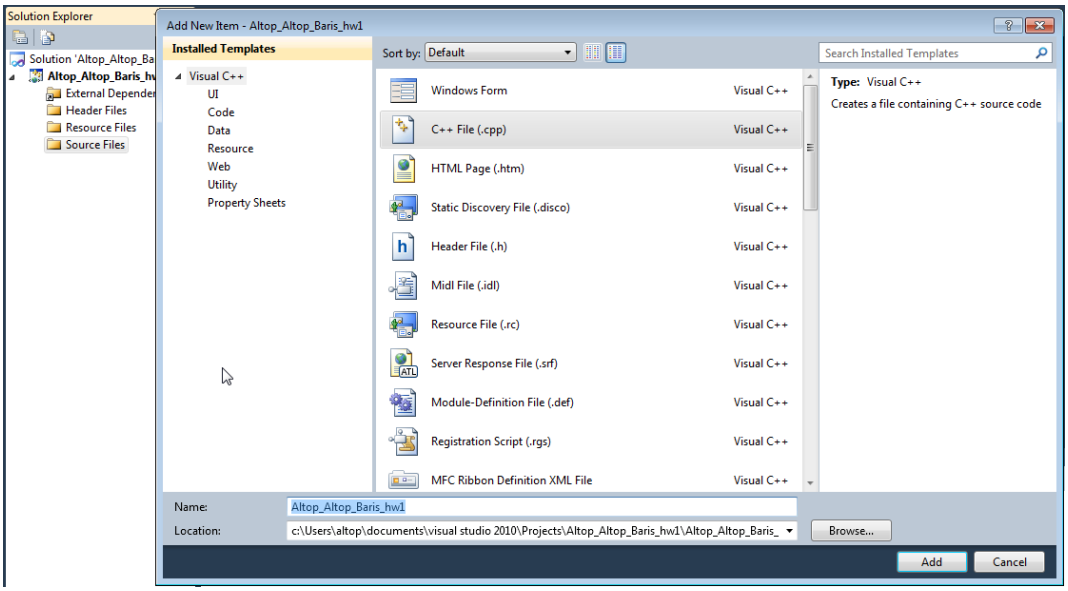
What and where to submit (PLEASE READ, IMPORTANT)?

You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework.

It'd be a good idea to write your name and last name in the program (as a comment line of course).

Submissions guidelines are below. Some 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.

- ❑ Name your cpp file that contains your program as follows.
  "**SUCourseUserName_YourLastname_YourName_HWnumber.cpp**"



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_hw5.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.
- ❑ Check that your compressed file opens up correctly and it contains your **cpp** file. You will receive no credits if your cpp file does not contain the correct file.
- ❑ The naming convention of the cpp file is the same as the cpp file (except the extension of the file of course). The name of the cpp file should be as follows.

"**SUCourseUserName_YourLastname_YourName_HWnumber.cpp**"

For example kipler_kipleroglu_zubeyir_hw5.cpp is a valid name, but hw4_hoz_HasanOz.cpp, HasanOzHoz.cpp are NOT valid names.

- ❑ 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 5 in the assignments list.
    3) Click on "Add Attachments" button.
    4) Click on "Browse" button and select the cpp file that you generated.
    5) Now, you have to see your cpp 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.
- ❑ 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 5 in the assignments list.
    3) Click on "Re-submit" button.
    4) Click on "Add/remove Attachments" button
    5) Remove the existing cpp file by clicking on "remove" link. This step is very important. If you do not delete the old cpp file, we receive both files and the old one may be graded.
    6) Click on "Browse" button and select the new cpp file that you want to resubmit.
    7) Now, you have to see your new cpp 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!

Hanefi Mercan and GülŞen Demiröz