CS201 – Spring 2021-2022
Take-Home Exam 5
– CS201 Course Student Grade Ranking with Class –
Due June 12th, Sunday, 23:55 (Sharp Deadline)

Brief Description

In this homework, you will modify THE-4 by adding a Student class as opposed to using multiple vectors. With this class you will be able to store all student information in a single vector.
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.

Your take-home exams will be automatically graded using GradeChecker, so it is very important to satisfy the exact same output given in the "Sample Runs". You can utilize GradeChecker (http://learnt.sabanciuniv.edu/GradeChecker/) to check whether your implementation is working in the expected way. To be able to use GradeChecker, you should upload all of your files used in the take-home exam. You should also submit all of your files to SUCourse (*your_main_cpp* file, and class you use like *student.cpp* for this take-home exam) **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_the5.cpp" (all lowercase letters). Please check the submission procedures of the take-home exam, which are listed at the end of this document.

For this take-home exam, your submissions will also be processed manually to check if the expected class functions are implemented or not. In case of missing member function implementations, you will lose points based on the severity of the deficiency. This **manual processing** does **not** mean that you may skip the format related issues; the **grade for the output of your program will still be obtained from the automated tool that we use for grading purposes**.

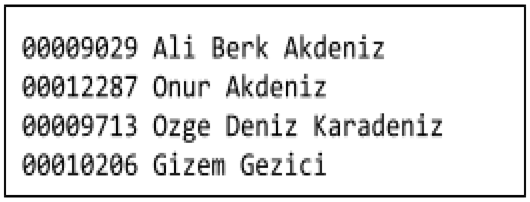
It is a **must** to use the given `student` class, any workaround will end up with a grade of zero (0) automatically.

You may use your codes from THE-4, **under the strict condition** that you implement the student class properly. As explained above, all submissions will be checked for proper implementation of the required class.

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 and last name of the student to separate these. 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:



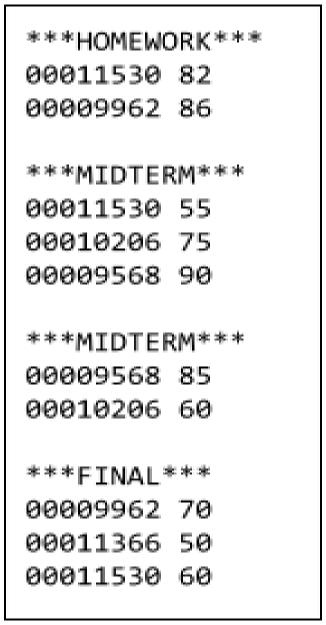
```
00009029 Ali Berk Akdeniz
00012287 Onur Akdeniz
00009713 Ozge Deniz Karadeniz
00010206 Gizem Gezici
```

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 contain 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 from 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 is in the 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 class data structure to encapsulate data of one student, such as student name, points, etc.

You can use a vector to store several students' data. Of course, the element type of this vector will be the previously defined student class. Here please remark that you CANNOT make any assumption about the number of students in the course. 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 student 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. <u>If it fails to open any of these files, then the program will terminate with an appropriate message (see sample runs for the message).</u>
- 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). <u>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</u>. 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 object 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 points value.
    - <u>Updating the data of a specific student</u>: 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 a descending manner.

- You may modify and use one of the 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. <u>If multiple students have both the same *point* and *last name*, then they should be in any order in the table</u>. 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.

Student Class

We will share the header file ***students.h*** for the class you will write. You must write all ten (10) of the member functions in the class in this header file, not one more or one less. Below you may find the details of the member functions:

Constructor

1. Default Constructor: In this default constructor you must assign an initial value to all private data members of the class.

Member Functions

1. setId: A member function to set the ID of a student. This function will get a string as its only parameter and set the **ID** *private data member* as the value of the parameter.

2. getId: Returns the ID. (Accessor)

3. setName : A member function to set the whole name of a student. This function will get a string as its only parameter and set the **firstName** and **lastName**, private data members, as the value of the parameter.

4. getFirstName : Returns first name, include middle name if exists. (Accessor)

5. getLastName : Returns last name. (Accessor)

6. setPoints : A member function to set the points of a student. This function will get a double as its only parameter and set the **points**, private data members, as the value of the parameter.

7. getPoints : Return points. (Accessor)

8. display : A member function to display a given rank, student's ID (**ID**), name (**firstName**), surname (**lastName**), and **point**, respectively. (Accessor)
This function will get a Student type vector which is a sorted vector based on the points of students, and an integer for the rank as its parameters.

9. setExamWeights : A member function to set the weights of exams. This function will get three doubles as its parameters and set the **hwWeight, midtermWeight** and **finalWeight**, private data members, as the value of the parameters.

10. getGrade : A member function calculates and returns the points. This function will get a string to the exam name and a double for the grade of the student for the homework, midterm or final exam as its parameters. Points will be calculated as a multiplication of grade and weight based on the exam.

You **must** add all of the member functions listed above to the given `student` class. You can also add free functions if you need them. For example, you might need another function to sort the vector.

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 to be 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. 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): 3
3, 00009568, Inanc Arin, 74.9
Enter the rank you want to query (enter 0 to exit): 2
2, 00009029, Ali Berk Akdeniz, 74.9
Enter the rank you want to query (enter 0 to exit): 5
5, 00009713, Ozge Deniz Karadeniz, 68.9
Enter the rank you want to query (enter 0 to exit): 9
9, 00010070, Batuhan Aydin, 63.9
Enter the rank you want to query (enter 0 to exit): 4
4, 00011464, Alper Batuhan Aydin, 74.7
Enter the rank you want to query (enter 0 to exit): 13
13, 00009962, Kubra Ak, 57.6
Enter the rank you want to query (enter 0 to exit): 16
Rank needs to be greater than 0 and smaller than 16!
Enter the rank you want to query (enter 0 to exit): 12
12, 00011366, Yigit Selim Akar, 58.6
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): 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): 3
3, 00009568, Inanc Arin, 75.6
Enter the rank you want to query (enter 0 to exit): 2
2, 00009029, Ali Berk Akdeniz, 78.4
```

```
Enter the rank you want to query (enter 0 to exit): 13
13, 00011366, Yigit Selim Akar, 54.6
Enter the rank you want to query (enter 0 to exit): -4
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 Selim 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): 0
Exiting...
```

General Rules and Guidelines about Homeworks

The following rules and guidelines will be applicable to all take-home exams, unless otherwise noted.

- How to get help?

You can use GradeChecker (http://learnt.sabanciuniv.edu/GradeChecker/) to check your expected grade. Just a reminder, you will see a character ¶ which refers to a newline in your expected output.

You may ask questions to TAs (Teaching Assistants) or LAs (Learning Assistants) of CS201. Office hours of TAs/LAs are at the course website.

- What and Where to Submit

You should prepare (or at least test) your program using MS Visual Studio 2012 C++ (Windows users) or using Xcode (macOS users).

It'd be a good idea to write your name and last name in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). If your name and last name is "İnanç Arın", and if you want to write it as comment; then you must type it as follows:

// Inanc Arin

Submission guidelines are below. Since the grading process will be automatic, students are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be 0.

- Name your submission file as follows:

    - 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 main cpp file that contains your program as follows: "**SUCourseUsername_THEnumber.cpp**"
    - Your SUCourse user name is actually your SUNet username, which is used for checking sabanciuniv emails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name (**use only lowercase letters**). For example, if your SUCourse username is "**altop**", then the file name should be: **altop_the5*.cpp*** (please only use lowercase letters).
    - Do not add any other character or phrase to the file name.
- Please make sure that this file is the latest version of your take-home exam program.
- Submit your work **through SUCourse only**! You can use GradeChecker only to see if your program can produce the correct outputs both in the correct order and in the correct format. It will not be considered as the official submission. You must submit your work to SUCourse. You will receive no credits if you submit by any other means (email, paper, etc.).
- If you would like to resubmit your work, you should first remove the existing file(s). This step is very important. If you do not delete the old file(s), we will receive both files and the old one may be graded.

- Grading, Review and Objections

Be careful about the automatic grading: Your programs will be graded using an automated system. Therefore, you should follow the guidelines on the input and output order. Moreover, you should also use the same text as given in the "Sample Runs" section. Otherwise, the automated grading process will fail for your take-home exam, and you may get a zero, or in the best scenario, you will lose points.

Grading:

- There is NO late submission. You need to submit your take-home exam before the deadline. Please be careful that SUCourse time and your computer time may have 1-2 minutes differences. You need to take this time difference into consideration.

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

- If your code does not work because of a syntax error, then we cannot grade it; and thus, your grade will be 0.

- Please submit your **own** work only. It is really easy to find "similar" programs!

- Plagiarism will not be tolerated. Please check our plagiarism policy given in the Syllabus or on the course website.

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 take-home exam from the email address provided in the comment section of your announced take-home exam grade or attend the specified objection hour in your grade announcement.

- Check the comment section in the take-home exam tab to see the problem with your take-home exam.
- Download the 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.

Good Luck!
E. Beyza Çandır & CS201 Instructors