

# CIS 331 Assignment #3

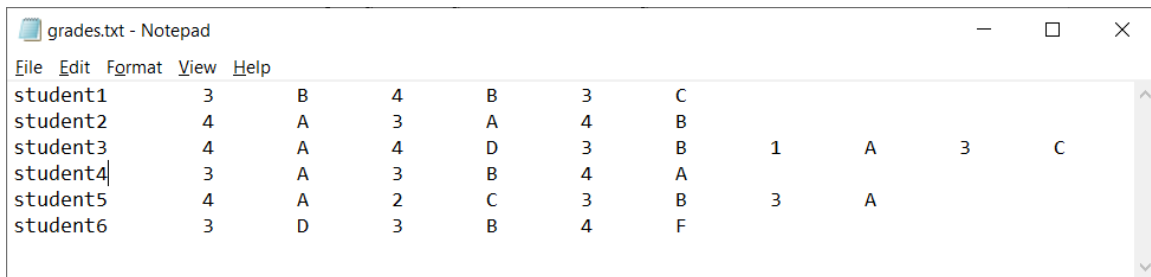
## Arrays and File I/O

### Student Grade Summary

For this assignment, you will write a program that reads student course and grade data from a text file, and then calculates GPAs and other information based on this file. The text file will contain one line of text for each student. Each line consists of the following data (delimited by tabs):

- 1) student
- 2) for each class the student takes, there are two values:
  - a. the number of credits in the class
  - b. the grade for the class (A, B, C, D, or F)

For example, consider this text file:



student1	3	B	4	B	3	C				
student2	4	A	3	A	4	B				
student3	4	A	4	D	3	B	1	A	3	C
student4	3	A	3	B	4	A				
student5	4	A	2	C	3	B	3	A		
student6	3	D	3	B	4	F				

The student called **student1** takes three classes: (3 credits of B, 4 credits of B, and 3 credits of C). As you can see, the student called **student3** take five classes. So, different students may take different numbers of classes.

Your program should first prompt the user to get the location of the input file. Then it will read the data from the file, put it in appropriate arrays (as described below), and output to the screen the following information: the student, total number of classes taken, total credit hours taken, and the GPA for that student.

During the process, the highest performing student will be identified and displayed, the average of all GPAs will be calculated, and then all students with a GPA higher than the average will be displayed.

When the program begins, you will see a prompt for the file path and the user should enter the value using the standard input stream (in Netbeans this is the output screen). After the user enters the input path of the file, the results for the data file shown above should look like this:

```
Enter full path of input file:
grades.txt
```

Students	#Class	#Cred	GPA
student1	3	10	2.70
student2	3	11	3.64
student3	5	15	2.60
student4	3	10	3.70
student5	4	12	3.42
student6	3	10	1.20

```
The top student is student4 with GPA 3.70
```

```
The average of all GPAs is 2.88
```

```
Students with higher than average GPA:
student2      3.64
student4      3.70
student5      3.42
BUILD SUCCESSFUL (total time: 9 seconds)
```

Note that the program first asks the user to enter the input file. For each student it accumulates the total number of courses, sums all the credits, and calculates the student's GPA based on summing each #credits X grade. A grade of A gives 4, B is 3, C is 2, D is 1, and F (or anything else) is 0.

Your program must be flexible enough to handle a variety of input files. There may be as many as 100 students in the file. But in all cases, a single line of the text file will be structured as described above.

Make sure to use good exception handling. If there is a runtime error you should display an appropriate message like this.

```
Enter full path of input file:
x
Error reading file
java.io.FileNotFoundException: x (The system cannot find the file specified)
BUILD SUCCESSFUL (total time: 2 seconds)
```

## Program Data Representation

To make this work, you will need to create four parallel arrays.

One array will contain the student. A second array will contain number of classes for each student. The third contains the total number of credits for each student. The fourth contains students GPAs. These are parallel arrays. This means that the same index in each array pertains to the same entity. For example, for the above data, the arrays will look like this after they are loaded:

Student		classes		credits		gpa
student1		3		10		2.70
student2		3		11		3.64
student3		5		15		2.60
student4		3		10		3.70
student5		4		12		3.42
student6		3		10		1.20

GPA does not need to be rounded in array, but it does need to be rounded when displayed in output.

These arrays must be large enough to handle a flexible number of students. Each should be the same size, and you should make it easy to change that size without doing multiple edits. This is a good use of a constant, which you can declare using the final keyword. T

Note that although these arrays must be large enough for the maximum amount, many of the elements of the arrays may be empty. So, you should also have a variable to keep track of how many students are actually read from the file. This **total students** variable will also be used to keep track of the next available element in the names and wages arrays, and can be used to index into these arrays. For example, the total students variable starts at 0, which means that you will index into element 0 of these arrays. For each student that you read from the file, you can increment the total students variable. Thus, after you have processed the first student, the **total students** variable will be 1, so you will index into element #1 of the arrays. After you've processed the second student from the file, the **total students** variable will be 2, so you will index into element #2 of the arrays. Thus, the total students variable serves two purposes: keeping a running count of the students and indexing into these parallel arrays.

You will also need other variables for keeping track of totals, inputting data from the file, finding the maximum value in the array, etc.

## Program Logic

The logic of the program works something like this:

- 1) Prompt the user for the name of the
- 2) Read the response from the user.
- 3) Open the input file specified by the user
- 4) Loop continuously:
  - a. Read the student name, place in array
  - b. Loop to read each class's credits and grade
    - i. Increment class count for student and store in array
    - ii. Accumulate sum of credits and store in array
    - iii. Calculate and sum score (credits times grade) for each class
  - c. Place calculated GPA into array
  - d. add one to the count of total students. You can use total student count to index into the arrays.
- 5) At this point you have accumulated all the data into the arrays. Now it is simply a matter of displaying the results, determining the max-GPA student, calculating the average GPA, and printing the students with higher than the average GPA. Each of these will be done in a loop.

## CAUTION

Conceptually, this is not a difficult problem, and doesn't take that much coding to accomplish. My solution contains a total of about 85 lines of code. The problem of reading from the console input, reading text files is simply what you learned from the class notes and examples of strings and stream processing. The techniques for processing the arrays are similar to what you learned from example 6.1 of the textbook. You can use the code from these examples to help you.

However, translating this from concept to result will be a challenging task. You do NOT want to put this off to the last minute. There is no doubt that you will encounter obstacles along the way.

Use the debugger to help identify problems and bugs. Also, make sure to use me as a resource. I will give guidance where appropriate. But most importantly, ***get an early start and work diligently and consistently through the whole week.***

## PROGRAM STYLE

In addition to correct functionality of the program, I expect you to adhere to sound programming style. Use descriptive variable names. Be sure to use proper indentation in your code and supply enough comments to make it clear what the program is doing. Use the naming conventions discussed in the textbook.

At the top of your program listing, you need to include comments with:

- 1) your name and student ID
- 2) the course and section
- 3) the assignment number
- 4) a statement assuring me that this work was done in accordance to the JMU Honor Code.

## DELIVERABLES

For this program, you will upload the following to Canvas:

- 1) The .java file of your solution code
- 2) A screen image (captured pasted into a Word document) showing the contents of memory after the data has been read into the arrays. This will require that you use the debugger. You should expand arrays to show the actual data in the individual elements.
- 3) A brief (1-2 sentence) description of what the debugger is showing you at the time of the breakpoint (also in the Word document)