EECS 211 – Spring Quarter, 2015
Programming Assignment 1

Due Friday, April 10th, 2015

In this assignment, you will write a program that averages exam scores for each student in a dataset. It will read the scores from a text file and write the results to the console.

The data will be formatted as follows:

- The data for each student will begin with the number of exam scores to be included in that set. This will be followed by that number of integers (each an exam score). For example, if the scores in one data set were 30, 40, and 50, the input would start with 3 followed by 30 40 50.
- At the end of each set of scores is an integer. If the integer is -1 it means there is at least one more set of scores (at least one more student); if it is -2 it means there is no more data.
- Each dataset has at least one student.

An example test data file to be used with this program is posted along with this document (along with the expected output). You should examine it to make sure that you understand what the format of the data is and also what to expect as output when your program is correct.

Your program should …

- Read in all of the scores for a student.
- Calculate the average of the scores for that student.
- Tally up the number of "invalid scores", that is scores outside the range 0-100.

If the value entered for a score is less than 0 or greater than 100, do not include that value in the average.

- Print "Student number: " followed by the student number (start counting at 1).
- If there are not ANY valid scores for a particular student, print the message "No valid scores for this student." and continue on to the next student.
- Otherwise print the average for that student and the number of invalid scores (in the format provided in the example below).
- Repeat this process for every student in the dataset.
- IMPORTANT – your output should match mine EXACTLY. I've provided the output for p1input.txt in a file called p1output.txt (note, this output was actually generated at the console, not written to a file). I recommend using a site like https://www.diffchecker.com/diff to verify that your output is exactly correct.

For example, if the input file contained the following data:

```
3  40  50  60  -1
0  -1
6  -1  0  1  99  100  101  -1
2  1000  2000  -2
```

(this is the contents of p1input.txt)

The console should output:

```
Student number: 1
Average: 50
Number of bad scores: 0

Student number: 2
No valid scores for this student.

Student number: 3
Average: 50
Number of bad scores: 2

Student number: 4
No valid scores for this student.

End of students.  Enter an integer to quit.
```
(this is the contents of p1output.txt)

Please note that in these examples, the average came out to a nice even 50, but the average could indeed be a decimal number. Given two scores to average, say 19 and 20, the average should be displayed at 19.5.

Name your file your-net-id.cpp (where letters in your netid appear in lowercase). For example, my file would be named sho533.cpp because sho533 is my netid.

Use the file assignment1starter.cpp as the basis for your work. Rename this file in the manner described above (e.g. sho533.cpp). Add your code, declarations, and comments to this file. Also download p1input.txt and p1output.txt.

NOTE: When the TAs grade your program or any of us help answer your questions, we will simply copy your .cpp file into a project that we have already created, rebuild the project, and run. If your files are not named exactly as indicated – your-net-id.cpp and p1input.txt – this process will not work and we will not be able to grade your work.

Submit your .cpp file to Canvas. (Submit only the .cpp file. Do NOT include any of the files generated by your C++ system.) If you upload code multiple times to canvas, we will grade the most recent one. Canvas may add numbers to the end of your filenames for these extra submissions, but don't worry, as long as the file you uploaded was named properly, all will be fine.

Comment and Suggestion:

In order to enable your Visual Studio or XCode project to have access to the external resource "p1input.txt," you'll have to tell the program where to look for this file. Specifically, you'll be telling your IDE where your "working directory" is (the folder that contains that .txt file). In XCode, you can do this by going to Product>Scheme>Edit Scheme, then Click on the options tab and set your custom working directory. In Visual

Studio, right click on the resource files folder in the solution explorer and add>new item...>Visual C++>Utility>Text File (.txt) and save it as p1input.txt. Then, simply copy-paste the text from the original p1input.txt file. Alternatively, google 'set visual studio working directory' and you'll find instructions.

We have provided a constant that can be used when you are developing your solution to determine whether to take input from the file or from the keyboard. When developing and testing software it is sometimes useful to be able to use the keyboard rather than have data come from a file. This allows the developer to rapidly test different scenarios without having to constantly edit a text file. In the case of this assignment, allowing input from the keyboard allows you to develop and test your code in a top-down manner, as described next.

It is rarely a good idea to try to develop an entire program in one step. It is much better to identify small steps that you can make in succession and do them one at a time. You implement the first small step and test it and fix any mistakes. When that much seems ok, add in the second step, and so on. In this assignment the top-down development approach breaks the assignment into several independent steps that are each quite easy.

1. Step 1: the outer loop. The outer loop proceeds until there are no more sets to process. Write an outer loop whose body simply prints a message saying that eventually this program will compute the average of a set of scores. Test this much with MODE set to 1 to see if you can correctly tell when to quit. In this case you will enter no counts or actual scores, just enter -1 and -2 to see if your outer loop correctly recognizes -2 and terminates.

2. The body of the outer loop processes one set of scores (i.e. – one student). This involves first reading the number of scores, then reading and processing the actual scores themselves, and finally computing the average or indicating no valid scores. You could implement this in stages:
    a. First, just read the count and see if you can read the correct number of scores. For this you could simply print out the number and the scores and thereby verify that reading is being done correctly. For this you could continue to use MODE set to 1 so that input comes from the keyboard or MODE set to 0 to see if you can read from the file.
    b. Next, add the code to actually compute the average – compute the sum of the scores and then divide. For this you could go back to MODE set to 1 so that you could use the keyboard and enter only valid scores and determine that your code for adding and dividing was correct.
    c. Finally, add in the tests for valid scores and for division by 0.