Homework

# 13

## Data Sets

This homework problem involves token processing along with the sentinel and data loops. Here's the specification.

> *Write a function named `dataSets()` whose argument is the name of a text file containing a sequence of data sets. Return a string containing the total and average in each data set.*

The text file contains a sequence of data entries where each entry is either:

- a **positive integer followed by a real number**, which indicates that the given real number occurs the given number of times in the data set, as in "`3 4.2`" which would indicate `3` occurrences of `4.2`
- the integer `0`, which indicates the end of the data set

You may assume that data sets are always well formed. If the input file does not exist, return the string "***`filename` cannot be opened.***" Consider the following data:

```
3 4.5 2 9.8 7 4.5 0 4 2.7 5 3.9 0 2 3.2 4 7.1 0
```

This input includes three data sets because it has three zeros.

- The first data set has `3` occurrences of `4.5`, `2` occurrences of `9.8` and `7` occurrences of `4.5` (terminated by a `0`).
- The second data set has `4` occurrences of `2.7` and `5` occurrences of `3.9`, again terminated by a `0`.
- The third set has `2` occurrences of `3.2` and `4` occurrences of `7.1`. It is also terminated by a `0`.

For the input above, the following `string` output should be produced:

```
"data set 1: total values = 12\naverage value = 5.3833\n\n"
"data set 2: total values = 9\naverage value = 3.3667\n\n"
"data set 3: total values = 6\n\naverage value = 5.8000\n"
```

Returned this as **a single string**. Note the blank line **between each data set**, but no blank line before the first or after the last. The value **0** that is used to terminate each data set should not be included in the computation.

Account for the multiple occurrences of each number in reporting the total number of values and the overall average for the data set. The first data set is specified by three pairs of numbers, but because of the duplicates, it contains a total of 12 numbers and the overall average of those 12 numbers is reported.

# Step 1 – Getting Started

In the **h13** folder.

1. Add the header guards, the prototype and documentation to **h13.h**.
2. Complete the file documentation in **h13.cpp**. Make sure you add your Canvas ID to the **STUDENT** variable.
3. Stub out the function and **make test**. The test program opens six different files, checking for both an empty file, a non-existent file, and four files that have random sequences inside them (testing the first two data sets).

To get some "points on the board", open the input file and return the error message as described above. That should give you about 11%.

# Step 2 – Read the Data Sets

Your code should look something like this (in pseudocode):

```
open filename as input file in
if in cannot be opened then
    return error message
```

Use **sentinel loops** to process the data sets. Each time through the loop, check for the end of that data set. Here's the logic for a sentinel loop to process only **one data set**:

```
read a number-to-repeat (int)
while the number-to-repeat is not 0 then
    read the value to process (double)
    add the number-to-repeat to the count
    add repeat * value to sum
    read the next number-to-repeat
add output to the result
```

Of course, you may need to process more than one data set. That means you'll need a loop for the whole file, and then, inside that, another loop for each data set.

Revised like this, your pseudocode becomes:

```
initialize data-set to 1
read repeat number (int)
while the input does not fail then (data loop)
    initialize sum, count
    while repeat is not 0 then
        read value, update count, sum (above)
        read next repeat
    add output to the result
    update data-set
    read next repeat
```

Notice the **read** (prime the loop), **test**, **read-next** from the **primed-loop pattern**. Note that for `in.fail()` to work, you must **first** attempt to read some data. There is nothing equivalent to Java's `hasNext()` methods.

## The Data Loop

Using the primed-loop pattern, the outer loop is **not** a **sentinel loop**; it is a **data loop** waiting for the **input device** to signal that it is **out of data**. The inner loop is a sentinel loop; it will continue while `repeat` is not `0`.

# Step 3: Performing the Output

**Think through** the output portion. Here's the output from earlier:

```
"data set 1: total values = 12\naverage value = 5.3833\n\n"
"data set 2: total values = 9\naverage value = 3.3667\n\n"
"data set 3: total values = 6\n\naverage value = 5.8000\n"
```

Every row has a newline at the end and the second and third rows have two newlines between them. (In other words, every pair of lines is separated by a blank line). Given this requirement, the pseudocode for output looks something like this:

```
initialize the result
. . .
if data-set > 1 then add "\n" to result
add  "data set " + data-set + ": total values = " + count + "\n"
   + "average value = " + (sum / count) + "\n" to result
. . .
return result
```

The **string** class has a function, **to_string()**, that will convert numbers into strings,. You can use that to implement the output.

Make sure you make **sum** a **double** and then **make test**. You'll be surprised to find that it doesn't solve the problem, because the numbers are not formatted.

For that, use a string stream.

## Formatting the Output

The ***ostringstream*** class works like ***ofstream*** except that the output is directed to a **string** rather than a file. This is exactly what you need.

Make sure you've added **<iomanip>** and **<sstream>** to your functions' headers, and then set the formatting on your **out** object. Instead of using concatenation, use regular formatted stream output. Then, instead of returning a **string** variable at the end of your function, return **out.str()**.

Do **make test** and, if you have 100%, **make submit**.

If there are still bugs, make sure you ask questions on Canvas or come by my office hours.