DVC Schedule v.1, Lab Assignment 4

Re-submit Assignment

Due Sep 27 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** cpp and h

Applying A Data Structure To A Big Data Application [DvcSchedule4.cpp *and* DynamicArray.h]

In this lab you will have your first experience with manipulating big data. The data is extracted from the DVC database of all class sections offered at DVC since the Fall 2001 semester. Your program is to list all of the subject codes (like COMSC, MATH, PHYS, etc), and include for each subject code the count of classes (e.g., MATH, 4514 classes).

Requirements. Write **DvcSchedule4.cpp** to read and parse the 70,000 line **dvc-schedule.txt** (https://drive.google.com/file/d/0B14YTZL55whCSGE0VjcyajZxSmc/view?usp=sharing) text file, and find each subject code in the file. Output each code to the console screen, in alphabetical order, with the number of classes offered under that code. Use your own **DynamicArray** template from Lab Assignment 3. Do *NOT* use any STL containers, and do *NOT* modify your H except to make corrections. Submit the H file, even if there are no corrections since lab 3. Canvas *will* add it's version tracking to the file's name -- that's okay.

NOTE: The **dvc-schedule.txt** file is expected to be in the *working folder*. In command line compiling, that's the same folder as the CPP and H files. In IDEs, you'll have to figure out for your IDE and project where is the working folder. Do *not* submit work that has a path designation in the file-open statement.

Note -- the **dvc-schedule.txt** file may contain duplicate entries. The combination of a term and section number is supposed to be unique. A duplicate entry is when the same term and section number pair appear in more than one record. Do NOT count duplicates -- skip them. That means to count a duplicate entry only once, ignoring all others. You'll need some way to track what's been counted so that you don't count the same section for the same semester more than once. When you are done processing the input file, output HOW MANY DUPLICATES you found and skipped in the input file. Check that number with your classmates, because you should all come up with the same number. You may use the Q&A section of this module for that.

You can expect the runtime to be several minutes. So that you don't stare at a blinking cursor while you wait for results, add a "progress bar". To do so, count the number of lines read from the file. For every 1000 lines read, output a dot -- like this:

```
cout << '.'; cout.flush( );
```

No **endl**. *You need* ***cout.flush( );*** to force output out of the output buffer and onto the console. After the EOF loop ends, output an **endl**, so that your output starts on a line *after* the line of dots. Or use some other method of indicating progress, as you prefer, but whatever you do, do *not* forget to flush! *Don't get this sent back for redo simply for forgetting this!*

Follow the algorithm developed in the lecture to solve this.

Be careful! Don't just accept whatever counts that your program gives you. Make sure that your program gives the right answers for the input file used. Try using a much shortened version of the TXT file, for which you know exactly what to expect. Also try loading the TXT file into Excel -- sort the data in column A, and count for yourself to verify the results of your app.

Example. (based on a previous year's version of the TXT file)

```
ADJUS, 557 sections
ADS, 206 sections
AET, 62 sections
ANTHR, 596 sections
ARABC, 13 sections
...
SPTUT, 12 sections
TAGLG, 8 sections
```

Hint

1. **Parse.** Start with the simple parsing code and see if you can read and parse the TXT file.
2. **Count.** Then add your code to count how many subject codes and how many sections of each.
3. **Duplicate Check.** Then add your code to track and exclude duplicates from counting.

If you do this right there should be a loop with a code block (or series of code blocks) that read and parse a line of input, then a code block (or series of code blocks) that check and reject duplicates, then a code block (or series of code blocks) that count. These three separate groups of code blocks *should NOT overlap*. They should each do their own job.

Lab Assignment Rubric

Criteria	Ratings							Pts
Fully accurate results, following all specifications **view longer description**	Works the first time. 70.0 pts	Works on the 2nd try 65.0 pts	Works on the 3rd try 60.0 pts	Works after 4 or more tries. 50.0 pts	Doesn't work after 2 weeks. Partial credit. 20.0 pts	Not submitted within two weeks of the due date. 0.0 pts	Work is not original -- appears to be a marked-up copy of the work of another or previous student. 0.0 pts	70.0 pts
Submits all work on time, fully complete if not fully correct. **view longer description**	Submitted on time 20.0 pts	Submitted on time, but one or more files are missing or not correctly named. 16.0 pts	Submitted on time, but with missing identification in one or more submitted CPP or H files. 15.0 pts	Submitted on time but not fully complete. 10.0 pts	Late or wholly incomplete! 0.0 pts			20.0 pts
Well-organized and professional quality code. **view longer description**	Fully meets expectations 10.0 pts	Mostly meets expectations, just needs to be a bit more careful. 8.0 pts	Many areas are well done, but there are a lot of areas that need work. 6.0 pts	Getting there, but needs to be a lot better. 3.0 pts	Needs a lot of work. See the instructor for guidance. 0.0 pts			10.0 pts
Total Points: 100.0								