Logistical Problems With Big Data, Reading

Memory

The first problem is memory. Remember that stack memory is (typically) 1MB by default, We have 70K records that we'll have to read in and store for processing. If each takes only 100 bytes, that's 7MB for an array. We'll have to deal with that.

There are two ways -- first, we could use that command-line compiler flag from the previous module to increase the stack memory for the program. Then we can use static arrays. Or we could use dynamic arrays that allocate in heap memory. Heap access is slower than stack access -- is that going to be a factor?

Loop Processing

We'll want to skip duplicate entries as we read the database file of DVC scheduling history. So we'll need a list where we can store the term+section combinations already seen, so if we see the same combination again, we can skip it. As we read the file a line at a time, that list size starts at zero, and ends at about 70K. On the average, 35K. So we have to read 70K records and do an average of 35K compares for each one to see if it's a duplicate or not. That's 2.45 *billion* compares. Is that going to be an issue? Or are 2GHz CPUs that can do 2 billion operations per second going to be able to handle that?

Before we produce output from whatever processing program we write, we'll want to order it -- probably alphabetically by course or subject code (like COMSC). How long will that list be? And can a loop of that size inside *another* loop of that size do nested for-loop sorting quickly enough?

Using Objects

We know we'll at least have to store term and section together -- maybe more. Should we devise an object for that with section and term as data members? Should it be a class with private data members and setters and getters, or is that too much overhead and we should use structs instead?

Validation

At the end of our program it will produce results (hopefully!). But how will we know if the results are correct? Will we accept them just because our program did something and did not crash and produced output? For example, if out program says that 21 sections of COMSC-210 were offered at DVC over the last 5 years, can we believe that answer just because it sounds reasonable, or is there a way to validate it. The TXT file is supposed to be openable in Excel. Do we know enough about using Excel to use it to answer the same questions our program answers, and validate it that way?

Data Integrity

How do we even know that our data set is any good? Well, you could say that it is our starting point, so by definition, it's good. But in reality, "spot checking" is always a good idea. For example, the published DVC catalogs from past years are the ultimate authority. But we cannot use it to count courses, because there may be a course in the catalog that's not in the schedule because it did not get scheduled. Printed schedules are not, real useful because sections get added and dropped after printing.

As you can tell, data integrity can be a really big deal. But for our application with the DVC schedule history, the only concern we'll deal with is the duplicate entry issue, because we *know* there are duplicates!

Our treatment of the duplicate entry issue will be as follows -- the *first* record we find with a unique term+section is the one that we'll treat as valid. We'll accept its course, instructor, and day/time/room. The *next* and any subsequent entries with the same term+section will simple be skipped. And just so we get an idea of how big the duplication issue is, we'll count how many duplicates got skipped, and report that somewhere in our programs' output.

Better Program Design

Experience and analytical methods will help us answer these very real concerns associated with program design. But for now we don't have much of either one when it comes to big data applications. As we go further into this course, you'll improve with your experience in doing lab assignments and in learning calculation techniques. Then in your career you'll continue to improve as you gain more experience with real-life solutions.