

BACKGROUND:

This project will be a simple linked list project. You will store student information in a list. The operations performed to the students will be the same as in project 2. You will add students, add courses to a particular student, update their QCAs, list students and courses for a particular student, and order the students based on their ID numbers.

DESIGN CONSIDERATIONS:

For this project you will create at least two classes to store the data. The first class will contain all the student information. There will need to be a field for each of the following: ID Number, Name, Major, Current QCA, Current Credits and a way to store the student's courses. You can store the student's courses in a stack, queue, plain dynamic array, or even a linked list. It is up to you. The second class will contain all the information about a course: CRN Number, Course Number, Course Department, Grade, and Quality Credits. All data files will be comma separated files.

You will read the entire file in at once and store all the information in memory. The file will start with a student's set of information followed by their course information. There will be a colon in the first column of the line to indicate that the first student's set of courses is complete and the next student's information is beginning. The next student's information will follow the same pattern as the first, student information followed by course information.

You can use the classes you already wrote and modify them, if necessary, to have the student's store their course information. You can also use the student and course classes that I wrote and posted to the course website. A caution, those classes are not fully tested and therefore may contain unknown bugs.

The student data file will be called "Student.data" and therefore can be opened as such and hard coded into your program. The command file will be called "Command.txt" and therefore can be opened as such and hard coded into your program.

The program will read a series of commands from the command file and produce output to the output file. The commands are: update, add, list, remove, and sort

update – this will recalculate QCA for every student that needs to have their QCA updated. For example, initially all students will need their QCAs updated with the new course information that was read from the file. This update will also include taking into account any prior QCA. After an add or remove command only one particular student will need their QCA updated. You will not automatically update anyone. You must wait for the update command.

add – this command has two flavors. add student and add course
add student will then be followed by the student's information. The student information will be the same as what is in the data file. The student information will not be followed by any course information.

add course will then be followed by the student's id number to which the course should be added and then the course information. The course information will be the same as in the data file. If the student is not present, then you cannot add the course.

list – this command will also have two flavors: list students and list courses

list students will instruct the program to list all the currently held students along with all of their information. This listing should be neatly organized in a table with a header row indicating the contents of each column.

list courses will require a student id number to be given and then the courses for that particular student will be listed. This listing should also be neatly organized in a table with a header row indicating the contents of each column.

remove – You guessed it this will also have two flavors: remove student and remove course

remove student will take an id number and remove the student from the in memory list of students. Since the student is gone, the student's courses will also go away.

remove course will take an id number and a CRN and will remove just that course from the student's course listing.

sort – This will sort the student records in an ascending sort based on their ID Number. I suggest an insertion sort. This time, I really mean it and it's not the hardest way. This is really a nice easy way and I will discuss it in class soon; so come to class. Of course, there are lots of ways to do this and the insertion sort is simply a suggestion.

Sample "Command.txt" file:

```
;just a sample input file to show the syntax of the commands
;update from the initial read
update
;add a student
add student 333-33-3333, Mike Major, Rel, 4.0, 60
;add a course to the newly added student
add course 333-33-3333 11011, 1001, Rel, C+, 3
;update that student
update
;list all the students and all their information
list students
;list the new student's courses
list courses 333-33-3333
;remove the new student's course
remove course 333-33-3333 11011
;remove the new student
remove student 333-33-3333
;this shouldn't do anything
update
;sort them
sort
;list the students in sorted order
list students
;were done!!
```

Also, after a remove course or add course, no automatic update of the QCA should take place.

To allow for debugging and possibly submission, I will set up the curator to allow you to test your program. This will mean that you will need to adhere to output standards. So for the listing the students follow the following format:

ID Number	Name	Major	QCA	Hours
123-45-6789	Joe E Student	CS	2.9733	30
234-23-5345	Sus E Student	Math	3.4822	45
134-54-3242	Harry McCracken	Phil	2.4466	15
333-33-3333	Mike Major	Rel	3.9190	63

For listing courses follow the following format:

ID Number	Name	Major	QCA	Hours	
333-33-3333	Mike Major	Rel	3.9190	63	
	CRN	CRS#	Dept	Grade	Hours
	11011	1001	Rel	C+	3
	12314	4512	CS	F	3
	13542	1235	Math	D	2

For the rest of the output messages follow the format shown in the sample output for Project 2. The only difference is in the listings. See this link for an example

<http://courses.cs.vt.edu/~cs1704/Spring03/P2out.txt>

Additionally, a design document like the one submitted for the first project is due on Wednesday, April 16, 2003 by 11PM. This design document should give the function interfaces and function calls. You should also represent the classes on this document as separate pieces. You do not need to indicate when the methods are invoke, only what the methods are.

Programming Considerations:

You must:

- have at least two classes, one for the students and one for the courses.
- have a linked list that is implemented in a class.
- echo any line that begins with a semicolon “;”
- Resubmit a final design as a separate piece.

Submitting your project

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically, **or** it will be archived for the TAs to grade later. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results

for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* and submission link can be found at: <http://www.cs.vt.edu/curator/>

Pledge

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:  
//  
// - I have not discussed the C++ language code in my program with  
//   anyone other than my instructor or the teaching assistants  
//   assigned to this course.  
//  
// - I have not used C++ language code obtained from another student,  
//   or any other unauthorized source, either modified or unmodified.  
//  
// - If any C++ language code or documentation used in my program  
//   was obtained from another source, such as a text book or course  
//   notes, that has been clearly noted with a proper citation in  
//   the comments of my program.  
//  
// - I have not designed this program in such a way as to defeat or  
//   interfere with the normal operation of the Curator System.  
//  
// <Student Name>
```

Failure to include this pledge in a submission is a violation of the Honor Code.