CSCI 13500. Programming Assignment 1.
DUE: 11:59pm Saturday May 7th.

This project is based on the first project. This time you must use vectors, structs, arrays and pointers. You will be graded for programming style on this assignment.

REVIEW: The Data with some changes

The two data files are *StudentData.tsv* and *HunterCourses.tsv*. Both files are tab-delimited (hence the *tsv* extension - tab separated values). And the data is structured so that you do not , actually you SHOULDN'T, have to use getline for your project.

StudentData.tsv **(*SD*)** contains student grade data, and **HunterCourses.tsv (*HC*)** contains data about individual courses.

SD contains 6,378 lines. Each line corresponds to a student taking a course. Here are some example lines:

```
20424297    1136   AFPRL        10200        A+     4
20424297    1139   CSCI         16000        W      -1
20424297    1142   PSYCH        18000        B      3
20424297    1142   PSYCH        22000        W      -1
20608974    1082   BLOCK        10000        XX     -1
20608974    1082   CHEM         102LC        B+     3.3
```

The first column is the student's CUNYFirst Empl ID, the second column is a code for the semester they took the course, the third column is the subject code, the fourth column is the catalog code (most times it is a number, but sometimes there are letters in the code), the fifth column is the letter grade received and the sixth column is the numeric equivalent of the letter grade.

The file is NOT sorted by Empl ID.

Some notes on the example lines:

- Student 20424297 took four courses AFPRL 10200, CSCI 16000, PSYCH 18000 and PSYCH 22000. Student 20608974 took two classes BLOCK 10000 and CHEM 102LC.
- You need to read all the data items, however you will only need to use the Empl ID, Subject, Catalog Number and Numeric Grade.
- When asked to calculate a GPA, numeric grades of -1 should be ignored.

HC contains 13,875 lines. Each line corresponds to an active course offering at Hunter College. Here are some example lines:

CSCI	12700	3.0	FSWR
CSCI	13500	3.0	RNL
CSCI	13600	2.0	RNL
PSYCH	22000	3.0	RLA
PSYCH	18000	3.0	RLA

The first column contains the subject, the second column contains the catalog number, the third column contains the number of contact hours and the fourth column contains a *designation code* which indicates whether a course offering is a Pathways course, a liberal arts course, a non-liberal arts course, etc.

NOTE: Not all courses in SD exist in HC. This happens when a course is renumbered or retired. For example a number of years ago CSCI 24500 was changed to CSCI 16000. If a student took CSCI 24500 before the number was changed, then CSCI 245 will exist in the SD file, but it won't be in the HC file.

The Project

Your project is going to use structs, vectors and linked lists. Use the following definitions:

```
struct Class
{
   int term;
   string subject;
   string catalog;
   string letGrade;
   double numGrade;
   Class* next ;
};

typedef Class* ClassList;

struct Student
{
   string    ID;
   ClassList ClassesTaken; // a linked list of Class objects
};

struct Course
{
   string subject;
   string catalog;
   double hours;
   string DR; // designation requirement
};
```

Rather than merging the two files and creating a new one, load each file into a vector. The StudentData file should be loaded into a vector of Student objects, and the HunterCourses file should be loaded into a vector of Course objects. **You should only go through each file once - without doing any calculations.** After your program has loaded both vectors, your program should generate exactly the same file as you created for your first assignment: Create a file called ***StudentSummary.tsv*** which will summarize the information in *StudentDataPlus.tsv* for each student. Each line should contain the student's EmplID, overall GPA, GPA in *CSCI* courses, and a percent of hours spent taking non-liberal arts courses. For this assignment, consider the three codes: *RNL*, *MNL*, and *GNL* as specifying non-liberal arts courses. All other designation codes are for liberal arts offerings.

Some style guidelines you should follow:

- **Use functions for everything!**
    - Your main block should only be 10-15 lines long, it could be even shorter
    - No single function should be longer than 10-15 lines long most should be shorter
    - Functions should perform one and only one task
    - If you first write pseudocode, then functions should follow naturally, for example:
        - Load student data into a vector
            - For each line in the file
                - find the student ID, create a class with data from the file, and add the class to the class list for the student
                - if the student isn't in the vector, then:
                    - etc.
                    - etc.

      You should have a function for loading the student data, a function to look in the vector for a student ID, a function to add a class to a class list, etc.

    - Since we haven't covered functions with stream parameters, here're two prototypes that you might want to use:

      ```
      void loadStudentData (ifstream& IN, vector<Student>& sVec);
      void loadCourseData  (ifstream& IN, vector<Course>&  cVec);
      ```

      Actually since C++ allows for overloading you could call both functions simply: `loadData`.

- **Comment everything!**
    - The top of your source file should have a long comment with your name, date submitted, class (CSCI 13500) and section number followed by a paragraph

about what the program does, what is the input, what format is the input in, what is the output, what format is the output in, etc. **w/o referencing any C++ terms**, followed by a paragraph about the key data structures you use in your program and how the program does what it does - in this paragraph you say things like: "The program uses structs to represent X, and linked lists to represent Y … The program first loads the data from xyz file into a vector of wqr objects. Then it does the same for the abc file and a vector of edf objects. Then …"

- Then at a minimum:

  - every function prototype needs a few lines of comments including a description of the formal parameters, what is calculated and returned (if anything) and the functions pre- and post-conditions.

  - every block of code should get at least a line of comment about what the block is doing.