CptS **122 – Data Structures**

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Programming Assignment 7: Attendance Tracker

III. Overview & Requirements:

Let us create an application that manages attendance for a course. This application has four major requirements:

Requirement 1 (Import records): The application must import records pertaining to each student registered for the course from a course list.
Requirement 2 (Mark absences): The application must allow the user to mark each student in the course as present or absent on any given day.
Requirement 3 (Generate reports): The application must generate reports based on criteria.
Requirement 4 (Menu): The application must support a user interface to the attendance tracker.

Import records: Records must be read from a comma-separated values (.csv) course file. A .csv file stores data as plaintext in tabular form. Each row in the file is considered a *record*. Each record consists of *fields* separated by commas. Please start with this .csv file. In this assignment the following fields will be present for each record:

- record number (max 3 digits)
- ID number (max 9 digits)
- name (last, first)
- email
- units (number of credits for class or AU for audit)
- program (major)
- level (freshman, sophomore, junior, senior, graduate)

You are required to use a dynamic singly linked list to store student records. As each record is imported from the file, the record must be inserted at the front of the list. Inserting at the front of a dynamic linked list is very efficient (constant time). You are required to implement two classes for the list. One class is the *node* class, which stores the fields acquired from each record. In addition to the fields in the file, you are required to add two extra fields in your node. These fields include *number* of absences and a *stack* (may be implemented using an array) for storing the dates of absences. The most recent absence date will always be at the top (Last-In First-Out, LIFO)! The second class is the *List* class, which is a *container* for the nodes. The List class will be considered your *master* list. You are required to implement only one class for the Stack. The Stack class will be implemented using an array. The Stack class must support push (), pop (), peek (), and isEmpty () operations. All of these should execute in constant time.

Mark absences: The user of the program should be able to view the *master* list of students in the course and mark absences for the current day. This may be implemented by simply traversing the linked list (linear time) and asking is the student absent? Yes or no? The date for the day must be derived from the computer's date. The following fragment of code illustrates how to derive the date from the computer:

```
// retrieved from stackoverflow - http://stackoverflow.com/questions/997946/how-to-get-current-time-and-date-in-c
time_t t = time(0);   // get time now
struct tm * now = localtime( & t );
cout << (now->tm_year + 1900) << '-'
    << (now->tm_mon + 1) << '-'
    << now->tm_mday
    << endl;
```

Generate reports: The user of the program should be able to generate two versions of reports. One version is a report that shows all of the students in the class and the number of times they have been absent, along with the date of the most recent absence (peek ()). A second version is a report that provides only the students and dates absent for those who are absent greater than some threshold set by the user. Write each report to a different .txt file.

Menu: At startup of the program a menu must be displayed. The menu must support six options. These include:

1. Import course list
2. Load master list
3. Store master list
4. Mark absences
5. BONUS: Edit absences

6. Generate report
7. Exit

Option 1: Reads the .csv course file and overwrites the master list.
Option 2: Populates the master list with previous nodes from master.txt.
Option 3: Stores the contents of the master list's nodes to master.txt.
Option 4: Runs through the master list, displays each student's name, and prompts if he/she was absent for the current day. The data must be pushed to the stack that is contained within the node representative of the student.
BONUS: Option 5: Prompts for an ID number or name of student to edit. Prompts for the date of absence to edit.
Option 6: Leads to submenu -> 1. Generate report for all students; showing only the most recent absence for each student. This is a peek () operation! 2. Generate report for students with absences that match or exceed (the number entered by the user).
Option 7: Exit the program.

You are required to define a class for your menu.