

CptS 122 - Data Structures



Programming Assignment 6: Attendance Tracker

Assigned: Wednesday, July 12, 2017

Due: Tuesday, July 18, 2017 by midnight

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- ✿ Design, implement and test classes in C++
- ✿ Declare and define *constructors*
- ✿ Declare and define *destructors*
- ✿ Compare and contrast *public* and *private* access specifiers in C++
- ✿ Describe what is an *attribute* or data member of a class
- ✿ Describe what is a *method* of a class
- ✿ Apply and implement *overloaded* functions
- ✿ Distinguish between pass-by-*value* and pass-by-*reference*
- ✿ Discuss *classes* versus *objects*
- ✿ Implement *container* classes
- ✿ Implement a *list* class
- ✿ Implement a *stack* class
- ✿ Read and write files in C++
- ✿ Programmatically acquire calendar dates

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- ✿ Analyze a basic set of requirements for a problem
- ✿ Compose basic C++ language programs
- ✿ Create basic test cases for a program
- ✿ Apply arrays, strings, and pointers

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.

BONUS: Edit absences - The user of the program should be able to access each student's record and edit absences. A search (linear time) through the master list based on student ID or name must be supported. If a student was initially marked absent for a date, but later was determined to be present, then the absence should be removed from the record. This includes updating the number of absences field. Be sure to add an Edit option to your menu!

IV. Submitting Assignments:

1. Using Blackboard Learn <https://learn.wsu.edu/webapps/login/> submit your assignment to your TA through the link ending with "-LEC". Under the "Content" link navigate to the "Programming Assignment Submissions" folder and upload your solutions to the appropriate "Assignment" space. You must upload your solutions as <your last name>_pa6.zip by the due date and time.
2. Your .zip file should contain a project workspace. Your project folder must have at least one header file (a .h file), two C source files (which must be .c files), and project workspace. Delete the debug folder before you zip your project folders.
3. Your project must build properly. The most points an assignment can receive if it does not build properly is 65 out of 100.

V. Grading Guidelines:

This assignment is worth 100 points. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

- 🐾 5 pts - Appropriate top-down design, style, and commenting according to class standards
- 🐾 25 pts - Appropriate design and implementation of Class *ListNode* and *List* (including member functions and data members)
- 🐾 15 pts - Appropriate design and implementation of Class *Stack* (including member functions and data members)
- 🐾 19 pts - Working "Import records" feature
- 🐾 12 pts - Working "Mark absences" feature
- 🐾 12 pts - Working "Generate reports" feature - each report is 6 pts
- 🐾 12 pts - Working "Menu" feature encapsulated by a menu object
- 🐾 **BONUS 20 pts** - Working "Edit absences" feature