COMP 2404 -- Assignment #2

Due: Friday, March 4, 2016 at 12:00 (noon)

Collaboration: You may work in groups of no more than two (2) students

Goal

You will modify your bookstore management program from either Assignment #1 or from the base code to store the courses in a linked list. You will also modify the "add course" feature, and add a "delete course" feature.

Learning Objectives

- gain additional experience with dynamically allocated memory in C++ using linked lists

Instructions:

1. Replace the `CourseArray` class with a new linked list class:

You will create a new linked list class called **`Dlist`**, which will hold the collection of courses as a linked list instead of an array. Courses will still be stored in alphabetical order by course code, which is the concatenation of the course subject and number. The linked list will be implemented as a **doubly** linked list based on the format that we saw in class, with each node holding a pointer to a **`Course`** object.

The **`Dlist`** class should:

- hold a pointer to the **head** of the list, but **not** contain a pointer to the tail
- provide an **`add`** function that takes a **`Course`** pointer and adds the course in its correct place in the list
- provide a **`remove`** function that takes a **`Course`** pointer and removes the corresponding course from the list
- provide additional functions as required (**hint**: a **`find`** function may be useful here)
- manage its memory to avoid memory leaks

Notes:

- **DO NOT use dummy nodes!** Every node in the list must correspond to a course.
- You will modify your program to use a **`Dlist`** object instead of a **`CourseArray`** object to hold the courses.
- All classes will continue to interact with the **`DList`** class the same way they did with the **`CourseArray`** class. The changes to the rest of the classes should be minimal.

2. Modify the "add course" feature

You will change the existing "add course" feature to add courses to the linked list in alphabetical order by course code.

3. Add a "delete course" feature

You will add a "delete course" feature that can be selected from the main menu:

- the user will be prompted to enter the course code for the course to be deleted
- your program will verify that this course exists
- the course will be removed from the linked list of courses
- deleting a course should automatically delete all the textbooks belonging to that course

4. Modify the "list courses" feature

You will change the existing "list all courses" feature so that it prints out the linked list of courses in increasing alphabetical order by course code, **and** by decreasing order. This is necessary in order to demonstrate the correct implementation of your doubly linked list class.

Printing out the linked list poses a new challenge in maintaining our existing design, specifically the separation of the UI and collection classes. We are **not** permitted to print out data from inside the collection class, since a collection class should not know how to interact with the outside world. Our solution here will be to implement two formatting functions on the **`Dlist`** class: one to create a string containing all the formatted course data in increasing order of course code, and the other to create a string with the same data in decreasing order. The UI class will invoke the formatting functions on the **`Dlist`** object, then output the resulting formatted string to the screen.

You will need to use the **`sstream`** library to convert an **`int`** (the course enrollment) to a **`string`**. The following code can serve as an example:

```
int num = 10;
string myStr;
stringstream ss;
ss << num;
myStr = ss.str();
```

Constraints

- your program must follow the existing design of the base code, including the encapsulation of control, UI, entity and array object functionality
- do not use any classes or containers from the C++ standard template library (STL)
- do **not** use any global variables or any global functions other than **`main`**
- do **not** use structs; use classes instead
- objects must always be passed by reference, not by value

Submission

You will submit in cuLearn, before the due date and time, the following:

- one **`tar`** file that includes:
  - all source and header files for your program
  - a Makefile
  - a readme file that includes:
    - a preamble (program and modifications authors, purpose, list of source/header/data files)
    - compilation, launching and operating instructions

If you are working with a partner:

- only **one partner** submits the assignment, and the other partner submits nothing; partners that make two separate submissions will be considered to have plagiarized each other's assignment and will be reported accordingly
- the readme file must contain the names of both partners
- the submitting partner must enter the names of both partners in the **Online Text** box of the *cuLearn* submission link

***** LATE ASSIGNMENTS WILL NOT BE ACCEPTED FOR ANY REASON *****

Grading

Marking components:

- Linked list class: 60%
  - 10 marks: correct representation as a doubly linked list
  - 5 marks: correct implementation of constructor
  - 5 marks: correct implementation of destructor
  - 15 marks: correct implementation of add function
  - 15 marks: correct implementation of remove function
  - 5 marks: correct implementation of formatting function with courses in increasing order
  - 5 marks: correct implementation of formatting function with courses in decreasing order
  - **Note**: Failure to print out the content of the linked list means that **none** of these 60 marks can be tested, which is an execution error under the deductions below and can result in a 50% deduction of the entire linked list marking component (30 marks)
- Add course feature: 10%
  - 10 marks: adds course in correct position
- Delete course feature: 30%
  - 5 marks: shows new option on the main menu
  - 5 marks: reads course code from user
  - 10 marks: finds correct course in list
  - 10 marks: removes course from list and correctly adjusts pointers

Deductions:

- Packaging errors:
  - 10 marks for missing Makefile
  - 5 marks for missing readme
  - 10 marks for **consistent** failure to correctly separate code into source and header files
- Major programming and design errors:
  - 50% of a marking component that uses global variables, global functions, or structs
  - 50% of a marking component that consistently fails to use correct design principles
  - 50% of a marking component that uses prohibited library classes or functions
  - 50% of a marking component where unauthorized changes have been made to provided code
- Execution errors:
  - 100% of a marking component that cannot be tested because the code does not compile or execute
  - 100% of a marking component that cannot be tested because the feature is not used in the code