COMP 2404 -- Assignment #3

<u>Due:</u> Friday, March 18, 2016 at 12:00 (noon)

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

Goal

You will modify your bookstore management program from either Assignment #2 or from the base code to implement overloaded operators on some classes and establish an inheritance hierarchy of bookstore products.

Learning Objectives

- practice implementing overloaded operators on different types of objects
- create a simple inheritance hierarchy

Instructions:

1. Overloaded operators

a. `CourseArray` or `Dlist` class

Depending on whether you start with the base code or with Assignment #2, you will implement the following operators for the class that holds the collection of courses (either **`CourseArray`** or **`Dlist`**):

Operator	*Parameter*	*Functionality*
`+=`	`Course& course` or `Course* course`	Add `course` to `this` collection
`+=`	`Dlist& courses` or `CourseArray& courses`	Add every element of `courses` to `this` collection
`-=`	`Course& course` or `Course* course`	Remove `course` from `this` collection
`-=`	`Dlist& courses` or `CourseArray& courses`	Remove every element of `courses` from `this` collection
`<<`		Format the collection of courses into a string containing all the formatted course data **Note**: this operator does not print to the screen! Usage: `outStr << courses;` where `outStr` is the formatted string and `courses` is the collection (either `CourseArray` or `Dlist`)

Notes:
- o All your overloaded operators must reuse existing functions everywhere possible.
- o You must enable cascading everywhere appropriate.

2. Testing the overloaded operators

To test the overloaded operators, your program must do the following. When the user chooses to add or remove a course, your program will:
- ask the user if they want to add/remove an individual course or several courses at once
- read from the user the course(s) they want to add/remove
- in case of one course, use the `+=` or `-=` operator that takes a `Course` reference as parameter
- in case of several courses, use the `+=` or `-=` operator that takes a `Dlist` or array reference as parameter

- use the << operator to format the courses collection before printing

Note: If you are starting from the base code, you will have to implement the remove course feature.

3. **Bookstore product hierarchy**

   a. **Design**

   You will come up with and implement a class hierarchy of products that the bookstore may want to sell in addition to textbooks. For example, they may sell computers, apparel, etc. Your class hierarchy must:

   - contain one base class
   - contain the **`Book`** class derived from the base class
   - contain at least **three** new kinds of products that make sense in the real world; no "dummy" classes!
   - each new class, including the base class, must have at least **two** data members that did not exist previously
   - use inheritance correctly

   Draw a UML class diagram to represent all the classes in the program (except the collection classes), including the new class hierarchy, all class attributes and operations, the associations between the classes (inheritance or composition), and the multiplicity and directionality of all associations. Do not show getter and setter functions, nor constructors or destructors.

   b. **Implementation**

   You will modify your bookstore management system to implement the class hierarchy that you designed in part (a). Your program will:

   - change the user interface so that every new kind of product in part (a) can be created dynamically, based on the user's wishes, and the objects must be initialized with user-specified values
   - store all products in **one** C-style array of pointers in the control object
   - add a "print inventory" feature that prints out the base class information for all the products currently for sale in the bookstore

   Note:

   o The comments section of each new class must explain the purpose of that class.

Constraints

- your program must follow the existing design of the base code, including the encapsulation of control, UI, entity and collection class 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:

- a UML class diagram (as a PDF file) that corresponds to the program design
- one **`tar`** file that includes:
  - o all source and header files for your program
  - o a Makefile
  - o a readme file that includes:
    - ▪ a preamble (program and modifications authors, purpose, list of source/header/data files)
    - ▪ compilation, launching and operating instructions

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

Grading

Marking components:

- Overloaded operators: 40%

7 marks:	correct implementation of += that takes a course object
10 marks:	correct implementation of += that takes a collection
8 marks:	correct implementation of -= that takes a course object
10 marks:	correct implementation of -= that takes a collection
5 marks:	correct implementation of << for formatting courses
Note:	Failure to call an operator from the test feature is an execution error under the deductions below and will result in a 100% deduction for that operator

- Test feature: 8%

2 marks:	reads course(s) to be added/removed from user
2 marks:	for one course, calls correct operator (1 mark for add, 1 mark for remove)
2 marks:	for several courses, builds temporary collection (1 mark for add, 1 mark for remove)
2 marks:	for several courses, calls correct operator (1 mark for add, 1 mark for remove)

- UML: 12%

12 marks:	presence of base class and 3 new derived classes that make sense (3 marks each), with correct associations

- Product hierarchy: 40%

16 marks:	presence of base class and 3 new derived classes that make sense (4 marks each)
8 marks:	presence of new data members that make sense (2 marks each class)
16 marks:	correct initialization of each new class based on user input (4 marks each)
Note:	Failure to allow the user to print the products to the screen is an execution error under the deductions below and can result in a 100% deduction for the entire product hierarchy component

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
  - 10 marks for bad style
- 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
  - 100% of a marking component that cannot be tested because data cannot be printed to the screen