COMP 2404 -- Assignment #4

Due: Tuesday, April 7, 2020 at 12:00 pm (noon)

Goal

For this assignment, you will write a C++ program to manage a group of students. You will also implement a class template, as well as some overloaded operators, and you will use exception handling techniques to deal with errors.

Learning Outcomes

With this assignment, you will:

- implement a class template and overloaded operators in C++
- use exception handling mechanisms to deal with errors

Instructions

1. Implement the `Object` class

You will create a new `Object` class that will serve as the base class for objects stored in a linked list, which will be implemented in one of the next steps. The `Object` class will contain the following:

(1) a data member for the unique identifier of an object; this will be represented as an integer, and it will be generated automatically by the constructor when an object is created

(2) a static data member called `nextId`, to store the identifier of the next object to be created
   (a) see the coding examples in section 3.1, program #6, for an example of how identifiers can be generated and assigned automatically using a static data member
   (b) make sure that the `nextId` member is initialized at file scope in the class source file

(3) a constructor that initializes the object identifier from the next available id (`nextId`) and increments this `nextId` for the next object to be created

(4) a getter function for the identifier

2. Modify the `Student` and `Date` classes

You will begin with the `Student` and `Date` classes that we worked on in the coding examples in section 3.1. You will use the `Student` class from program #7, and the `Date` class from program #3.

(1) You will modify the `Student` class as follows:
   (a) change the `Student` class so that it is derived from the `Object` class
   (b) replace the `print()` function with the overloaded stream insertion operator (`<<`)
   (c) write the overloaded less-than operator (`<`) so that the left-hand side student is considered the lesser if its name comes first in alphabetical order
   (d) write the overloaded equality operator (`==`) so that both students are equal if their names and their student numbers are equal

(2) You will modify the `Date` class as follows:
   (a) change the `Date` class so that it is derived from the `Object` class
   (b) replace the `printShort()` function with the overloaded stream insertion operator (`<<`)
   (c) implement the overloaded less-than (`<`) operator that compares two dates (think about the algorithm required here)
   (d) implement the overloaded equality (`==`) operator that compares two dates

3. Modify the `List` class

You will begin with the `List` class that we worked on in the coding examples in section 3.1, program #7, and you will make it a class template. You will also modify the class so that it stores data as a doubly linked list. You will make the following changes:

(1) make the `List` class into a class template; to ensure consistency, the class template will only store *pointers*, so the list's nodes should store a `T` **pointer** as data; you will need to modify the member functions accordingly

(2) modify the `add()` function so that it inserts new elements in increasing order; you must use one of the overloaded comparison operators that you implemented for the `Student` and `Date` classes; remember: you **must** dereference the pointers to compare the actual objects

(3) modify the `add()` function so that it does *not* allow duplicate data; you must use one of the overloaded comparison operators that you implemented for the `Student` and `Date` classes

(4) modify the `del()` member function so that it takes an integer identifier as parameter; since our data will all be derived from the `Object` class, we can use the object identifier to find the object to be removed from the list; the memory for the removed object should be deallocated; you can remove the `del()` function's second parameter

(5) change the list into a doubly linked list; this will require changes to the class definition, as well as the `add()` and `del()` member functions; the `print()` function must print the data once in the forward direction, then a second time in the backward direction, so that the links are thoroughly tested

(6) modify the destructor so that it deallocates the list's data, as well as the nodes

(7) replace the `add()` member function with the overloaded `+=` operator; you must enable cascading

(8) replace the `del()` member function with the overloaded `-=` operator; you must enable cascading

4. Implement the `Control` class

You will create a new `Control` class that implements the control flow for the entire program. The `Control` class will use the skeleton `launch()` member function provided for you in the `a4-posted.tar` file available in *cuLearn*. The `Control` class must also contain the following:

(1) a data member for a collection of `Student` object pointers; this must be represented using the `List` class template implemented in the previous step

(2) a data member for a collection of `Date` object pointers; this must also use the `List` class template

(3) a data member for the `View` object that will be responsible for most user I/O
   (a) the `View` class is provided for you in the `a4-posted.tar` file

(4) a `void initStudents()` member function that initializes the students of this program; this function will do the following:
   (a) dynamically create at least 10 student objects, without any duplicate information
   (b) add each student object to the student collection using the overloaded `+=` operator
      - the ordering must test the operator thoroughly

(5) a `void initDates()` member function that initializes the dates of this program; this function will do the following:
   (a) dynamically create at least 10 date objects, without any duplicate information
   (b) add each date object to the date collection using the `+=` overloaded operator
      - the ordering must test the operator thoroughly

(6) a `launch()` member function that implements the program control flow and does the following:
   (a) call the `initStudents()` and `initDates()` member functions to initialize the data members
   (b) use the `View` object to display the main menu and read the user's selection, until the user chooses to exit

(c) if the user chooses to create a new student:
- use the `View` object to read from the user the data for the student to be created
- dynamically allocate the new `Student` object
- add the new object to the student collection using an overloaded operator

(d) if the user chooses to remove a student:
- use the `View` object to read from the user the object id of the student
- remove the student from the student collection using an overloaded operator

(e) if the user chooses to print the students, call the appropriate function

(f) once the user chooses to exit, print the content of both collections (students and dates) to the screen

NOTE: A skeleton `launch()` function is provided for you and can be downloaded from *cuLearn* in the `a4-posted.tar` file. If you use this skeleton function, you must add the code required so that the function performs all the tasks described above.

5. Using exception handling

You will use exception handling to deal with the case where the user chooses to remove a student with an identifier that is not in the student collection. The exception must be thrown from the `List` class and handled in the `Control` class by using the `View` object to print an error message to the screen.

6. Write the `main()` function

Your `main()` function will declare a `Control` object and call its `launch()` function. The entire program control flow must be implemented in the `Control` object as described in the previous instructions, and the `main()` function must do nothing else.

7. Test the program

(1) make sure that the data you provide exercises all your functions thoroughly; failure to do this will result in **major deductions**, even if the program appears to be working correctly

(2) check that the student and date information is correct when it is printed at the end of the program

(3) make sure that all dynamically allocated memory is explicitly deallocated when it is no longer used; use `valgrind` to check for memory leaks

Constraints

1. your program must follow correct encapsulation principles, including the separation of control, UI, entity, and collection object functionality
2. do not use any classes, containers, or algorithms from the C++ standard template library (STL)
3. do not use any global variables or any global functions other than `main()`
4. do not use structs; use classes instead
5. objects must always be passed by reference, never by value
6. functions must return data using parameters, not using return values, except for getter functions
7. existing functions must be reused everywhere possible
8. all basic error checking must be performed
9. all dynamically allocated memory must be explicitly deallocated
10. your classes must be thoroughly documented in every class definition, as discussed in the course material, section 1.3

Submission

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

- one **`tar`** or **`zip`** file that includes:
  - all source and header files, including the code provided
  - a Makefile
  - a README file that includes:
    - a preamble (program and revision authors, purpose, list of source/header/data files)
    - compilation and launching instructions

NOTE: Do **not** include object files, executables, duplicate files, or unused files in your submission.

Grading (out of 100)

Marking components:

- 6 marks: correct implementation of `Object` class
- 20 marks: correct implementation of `Student` and `Date` classes
- 44 marks: correct implementation of `List` class
- 30 marks: correct implementation of `Control` class

Execution requirements:

- all marking components must be called, and they must execute successfully to receive marks
- all data handled must be printed to the screen for marking components to receive marks

Deductions:

1. Packaging errors:
   (1) 10 marks for missing Makefile
   (2) 5 marks for a missing README
   (3) 10 marks for consistent failure to correctly separate code into source and header files
   (4) up to 10 marks for bad style or missing documentation
2. Major programming and design errors
   (1) 50% of a marking component that uses global variables, or structs
   (2) 50% of a marking component that consistently fails to use correct design principles
   (3) 50% of a marking component that uses prohibited library classes or functions
   (4) 100% of a marking component that is *replaced* by prohibited library classes or functions
   (5) 50% of a marking component where unauthorized changes have been made to the provided code
   (6) up to 100% of a marking component where Constraints listed are not followed
   (7) up to 10 marks for memory leaks
   (8) up to 10 marks for bad style
3. Execution errors
   (1) 100% of a marking component that cannot be tested because it doesn't compile or execute in VM
   (2) 100% of a marking component that cannot be tested because the feature is not used in the code
   (3) 100% of a marking component that cannot be tested because data cannot be printed to the screen
   (4) 100% of a marking component that cannot be tested because insufficient datafill is provided