SYDE 121 – Digital Computation
Lab #10

Exercise 1: Improving the Rational class

Learning Objectives: Implement and use overloaded operators.

Read This First

Use the Rational class you created in Lab #9 as a basis for this lab.

What to Do

In Lab #9, you were asked to create a class to create and use rational numbers. Here, you are asked to use operator overloading to implement the same functionality and add some new functionality.

- Overload +, -, *, / mathematical operators.
- Overload >, <, >=, <=, = =, ! = logical operators.
- Overload >> and << operators. For output, if the numerator is zero, just display the value 0. If the denominator is 1, only display the numerator. Otherwise, display the value in fractional form e.g., 3 / 5.

In order to evaluate your new class, download from the course website the driver function that tests all of the functionality. You should verify that your class works correctly for a variety of inputs.

What To Hand In

Hand in to the submission box and email to the course account your class definition and implementation.

Exercise 2: Creating an Inherited Class

Learning Objectives: Learn how to create an inherited class and implement relevant functionality.

Read This First

Recall the "university" hierarchy that we studied during lectures. You will be asked to extend the hierarchy by adding a class 'Undergrad'. This new class will have Student as its parent class.

Download the necessary files for the class hierarchy (i.e., Person, Student, and Date classes) from the course website and the main function (not really a driver function since not all functionality is tested).

What to Do

Add the Undergrad class as a child of the Student class. The Undergrad class will contain private members for: grades (int array of NUM_COURSES (set to 5)), a status (an `enum` type with values PASS, FAIL, and UNDECIDED), current year of the student (1, 2, 3, or 4), and term (A or B).

Provide the following functionality to the Undergrad class:

- default constructor (should initialize grades array to zero and set status)
- constructor with arguments for the current year and term (should allow user to set grades array and then have program determine the status)
- constructor with arguments for current year and term, `Student`, and `Person` (should allow user to set grades array and then have program determine the status) Note: pass arguments to `Student` constructor that will in turn call the `Person` constructor.
- constructor with arguments for itself (current year, term), `Student`, `Person`, and `dateofbirth` (should initialize grades array and set status) Note: pass arguments to `Student` constructor that will in turn call the `Person` constructor.
- default destructor
- `'enter_grades'` – should be called from non-default constructors (do not worry about checking for valid grades)
- `'print_grades'`- self-explanatory
- `'print'` – should print all data associated with the undergrad (including all member data from parent classes)
- `'determine_status'` – determine status (ie. set the status variable) based on the average of the grades (set a PASSMARK of 50)
- `'get_status'` – return status (no need to set status since determined by grades i.e. we will not override the status)
- get/set functions for current year and term (make sure that the entered year is set to 1, 2, 3, or 4 and that the entered term is either 'A' or 'B'. If incorrect, give user endless opportunities to get it right!)
- '`age_at_convocation`' – assume that convocation takes place on May 15 each year. Determine the age of the student on this day (ignore the possibility that they failed the term). Pass no arguments to this function and return an int indicating the age (in years).

Add functionality to the main program to demonstrate the aspects of your new class. Do not modify any of the provided classes (`Person`, `Date`, and `Student`). Make sure that you demonstrate that the `Undergrad` class is able to access members of parent classes by calling base class members using an Undergrad object.

What To Hand In

Only the main.cpp, undergrad.cpp, and undergrad.h are required to be submitted. Do not submit the other components of the hierarchy since they are not to be modified.

Exercise 3: Classes and Dynamic Memory Allocation

Learning Objectives: Create a copy constructor, destructor, and overloaded assignment operator for a class that uses dynamic allocation.

Read This First

Download the Array class (array.cpp, array.h, and main_array.cpp) from the course website. The Array class uses dynamic memory allocation to create an integer array. This code does not work – you have to provide the rest of the functionality.

What to Do

To help you along and to allow you to focus on the learning objectives, most of the code has been written for you. All you have to do is provide the function bodies. In array.cpp you will find occurrences of the following statement:

```
// provide function body
```

You have to provide the code for each of these occurrences. There is no need to modify the main.cpp or array.h, although you should (as usual) understand how these are used.

To minimize your time requirements, do not bother documenting each of the member functions. However, do add the standard header, as usual.

What To Hand In

Hand in to the submission box and email to the course account the array.cpp file.

Due Date

Two lab sessions are set aside to complete this lab. As a result, the due date is Monday December 5 at 9am.