SYDE 121 – Digital Computation
Lab Number 9

Exercise 1: Improved Date Class

Learning Objectives: To practice programming with classes and objects.

Read This First

Dates are a common application for the introduction of objects as implemented in C++ using classes. In this exercise you will use and add functionality to a predefined Date class. An important capability of this Date class is its ability to handle the concepts of "valid" and "invalid" dates. For example, if a Date object is created with nonsensical input, then that object should be invalid.

What to Do

Set up a project called UseDate. Copy in the files from the website and create a project.

Read the Date class declaration in the file Date.h. Read the comments describing the purpose of each member function. Then read the main function (in UseDate.cpp), and run the program a few times to see if it behaves the way you expect.

Now add a new member function to the Date class. The function should be called **backup**; its effect should be to step a Date object back one day. (In other words, it should do the opposite of the **advance** function.)

Since the following block of code is called many times, replace all instances with a new member function call called 'check( )':

```
if ( ! ( is_valid() ) )
{
      cout << "Not valid. Exitting." << endl;
      exit(1);
}
```

Get rid of the original main( ) function. Write a new main( ) function that asks the user for the date of an airline flight and the number of days in advance that the ticket must be purchased; the user should then be advised of the last possible day for buying the ticket. A sample dialog might go like this:

```
What is the date of the flight?  1996 9 27
The date read was 1996.09.27
How many days earlier must the ticket be bought? 14
Doing a computation for a 14-day advance purchase...
You must buy the ticket on or before 1996.09.13
```

Test the completed program by running it several times. Make sure you try cases where you must back up to the previous month, and cases where you must back up to the previous year.

Background: The Modern Calendar

The calendar we use today is called the *Gregorian calendar*. In this calendar, a year is a leap year if and only if it falls into one of the two following categories:

- multiples of four that are not multiples of 100; or
- multiples of 400.

For example, 1996 was a leap year because it is in the first category, and 2000 was a leap year because it is in the second category. On the other hand, 1900 was not a leap year, and 2100 will not be a leap year either.

The Gregorian calendar was invented in 1582 as a replacement for the *Julian calendar*, in which every fourth year was a leap year, without any exceptions. Different countries adopted the Gregorian calendar in different years. Switching systems required a country to drop some days from its calendar in the year of the switch. England and its colonies changed calendars in 1753, dropping eleven days from the 1752 calendar.

So if for some reason you are writing a program that works with dates far back in history, you may have quite a messy programming problem. However, in this exercise, you need only concern yourself with dates from 1753.01.01 onwards, i.e., modern dates.

What To Hand In

Hand in and email the revised Date.h, Date.cpp and main program. Make sure that you (a) use the proper header information in your email and (b) submit the hardcopy to the proper box.

Exercise 2: Create a `Rational` class

Learning Objectives: To develop a class of your own. To learn about driver programs.

Read This First

You are expected to create your own class from scratch. The class will be tested with a driver program that is provided.

What to Do

Download the driver program from the course website. A driver program is a tool used by a programmer to test the functionality of a set of code. In this case, you have been provided with a driver program that tests the Rational class you will be asked to develop.

Create a class called **`Rational`** for performing arithmetic with fractions. Use integer variables to represent the private data of the class – the numerator and denominator. Provide a constructor function that enables an object of this class to be initialized when it is declared. The constructor should contain default values in case no initializers are provided and should also always store the fraction in reduced form (eg. 2/4 should be stored as 1/2). Keep the reducer member function as private (why should you do this? This is an important question - if you don't know the answer, ask for assistance). Provide public member functions for each of the following (for each of (a) through (d) the result should be stored in reduced form):

a) Addition of two **`Rational`** numbers.

b) Subtraction of two **`Rational`** numbers.

c) Multiplication of two **`Rational`** numbers.

d) Division of two **`Rational`** numbers.

e) Printing **`Rational`** numbers in the form a/b.

f) Printing **`Rational`** numbers in floating point form.

Hint

Make sure you develop solid code for this problem because you will be using it in Lab #10. Remember to properly document your code.

What To Hand In

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

Due Date

Monday, November 21 by 9:00am.