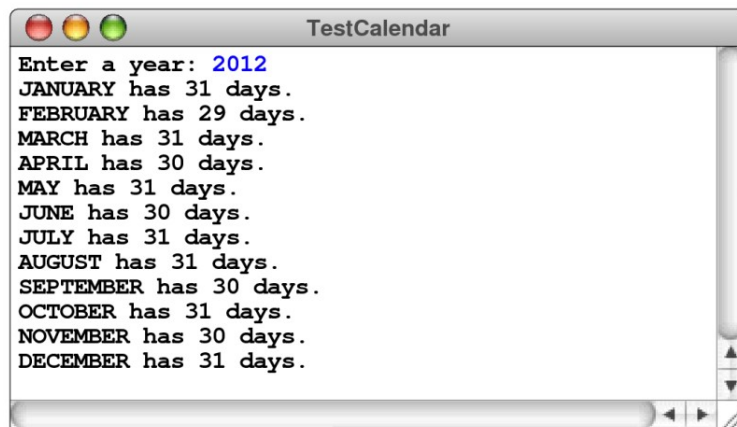


## CSC3002 Assignment 2

### Problem 1

#### Part1: (Exercise 2.11)(1.25')

Using the *direction.h* interface [Figure 2-7 in textbook] as an example, design and implement a *calendar.h* interface that exports the **Month** type from Chapter 1, along with the functions **daysInMonth** and **isLeapYear**, which also appear in that chapter. Your interface should also export a **monthToString** function that returns the constant name for a value of type **Month**. Test your implementation by writing a main program that asks the user to enter a year and then writes out the number of days in each month of that year, as in the following sample run:



```
TestCalendar
Enter a year: 2012
JANUARY has 31 days.
FEBRUARY has 29 days.
MARCH has 31 days.
APRIL has 30 days.
MAY has 31 days.
JUNE has 30 days.
JULY has 31 days.
AUGUST has 31 days.
SEPTEMBER has 30 days.
OCTOBER has 31 days.
NOVEMBER has 30 days.
DECEMBER has 31 days.
```

#### Part2: (Exercise 6.5)(1.25')

Extend the *calendar.h* interface above, so that it also exports a **Date** class that exports the following methods:

- A default constructor that sets the date to January 1,1900.
- A constructor that takes a month, day, and year and initializes the **Date** to contain those values. For example, the declaration

**Date moontanding (JULY, 20,1969);**

should initialize **moonLanding** so that it represents July 20,1969.

- An overloaded version of the constructor that takes the first two parameters in the opposite order, for the benefit of clients in other parts of the world. This change allows the declaration of **moontanding** to be written as

**Date moonlanding (20, JULY, 1969);**

- The getter methods **getDay**, **getMonth**, and **getYear**.
- A **tostring** method that returns the date in the form  $dd - mmm - yyyy$ , where  $dd$  is a one- or two-digit date,  $mmm$  is the three-letter English abbreviation for the month, and  $yyyy$  is the four-digit year. Thus, calling **tostring(moonlanding)** should return the string **"20-Jul-1969"**.

**Part3:** (Exercise 6.6)(2.5')

Extend the *calendar.h* interface still further by adding overloaded versions of the following operators:

- The insertion operator  $<<$ .
- The relational operators  $==$ ,  $!=$ ,  $<$ ,  $<=$ ,  $>$ , and  $>=$
- The expression  $\text{date} + n$ , which returns the date  $n$  days after date
- The expression  $\text{date} - n$ , which returns the date  $n$  days before date
- The expression  $d_1 - d_2$ , which returns how many days separate  $d_1$  and  $d_2$
- The shorthand assignment operators  $+=$  and  $-=$  with an integer on the right
- The  $++$  and  $--$  operators in both their prefix and suffix form.

Suppose, for example, that you have made the following definitions:

**Date electionDay(6, NOVEMBER, 2012);**

**Date inaugurationDay(21, JANUARY, 2013);**

Given these values of the variables, **electionDay < inaugurationDay** is true because **electionday** comes before **inaugurationDay**. Evaluating **inaugurationDay - electionday** returns 76, which is the number of days between the two events. The definitions of these operators, moreover, allow you to write a **for loop** like

**for (Date d = electionDay; d ≤ inaugurationDay; d++)**

that cycles through each of these days, including both endpoints.

**Requirments & Hints:**

For this problem, the three parts are combined together. The header file is in *calendar.h*; the source file is in *calendar.cpp*; the test part is in *Assignment2.cpp*. You need to finish all **TODO** parts in these files. The given output samples are shown in the comment of *Assignment2.cpp*.

## Problem 2

(Exercise 5.13)(2.5')

*And the first one now*

*Will later be last*

*For the times they are a-changin'.*

*-Bob Dylan, "The Times They Are a-Changin'," 1963*

Following the inspiration from Bob Dylan's song (which is itself inspired by Matthew 19:30), write a function

```
void reversequeue (Queue<string> & queue);
```

that reverses the elements in the queue. Remember that you have no access to the internal representation of the queue and must therefore come up with an algorithm—presumably involving other structures—that accomplishes the task.

### Requirments & Hints:

Please fill in the **TODO** part of *reversequeue.h* and *reversequeue.cpp*. The test part is located in *Assignment2.cpp*.

## Problem 3

(Exercise 5.19)(2.5')

In May of 1844, Samuel F. B. Morse sent the message "What hath God wrought!" by telegraph from Washington to Baltimore, heralding the beginning of the age of electronic communication. To make it possible to communicate information using only the presence or absence of a single tone, Morse designed a coding system in which letters and other symbols are represented as coded sequences of short and long tones, traditionally called dots and dashes. In Morse code, the 26 letters of the alphabet are represented by the codes shown in Figure .

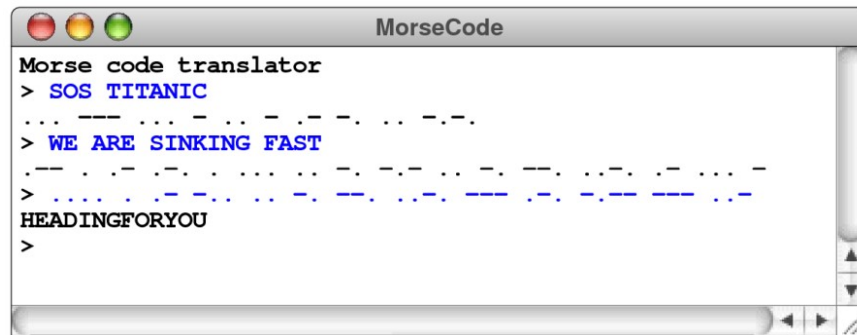
A	• —	H	• • • •	O	— — —	V	• • • —
B	— • • •	I	• •	P	• — — •	W	• — —
C	— • • • •	J	• — — —	Q	— — • —	X	— • • —
D	— • •	K	— • —	R	• — •	Y	— • — —
E	•	L	• — • •	S	• • •	Z	— — • •
F	• • — •	M	— —	T	—		
G	— — •	N	— •	U	• • —		

Write a program that reads in lines from the user and translates each line either to or from Morse code depending on the first character of the line:

- If the line starts with a letter, you want to translate it to Morse code. Any characters other than the 26 letters should simply be ignored.

- If the line starts with a period (dot) or a hyphen (dash), it should be read as a series of Morse code characters that you need to translate back to letters. Each sequence of dots and dashes is separated by spaces, but any other characters should be ignored. Because there is no encoding for the space between words, the characters of the translated message will be run together when your program translates in this direction.

The program should end when the user enters a blank line. A sample run of this program (taken from the messages between the Titanic and the Carpathia in 1912) might look like this:



### Requirments & Hints:

A sample of creating morse code map is given in *morsecode.cpp*. You needn't to implement it by yourself. For inverted map, it is not allowed to generate inverted map manually (typed inverted map directly). You need create the inverted map by using the created map before. Please fill in the **TODO** part of *morsecode.h* and *morsecode.cpp*. The test part is located in *Assignment2.cpp*.

## Requirements for Assignment

We provided the header file *calendar.h*, *reversequeue.h* and *morsecode.h*; source file *calendar.cpp*, *reversequeue.cpp* and *morsecode.cpp*; the test file *Assignment2.cpp*. You need copy these files under the **src** folder in the Stanford Library and delete the existed *hello.h* and *hello.cpp*. Please rename the project name as your student ID. Then, you can write and debug your codes in all TODO parts.

Finally, please pack your **whole project folder into a single .zip file**, name it using your student ID (e.g. if your student ID is 123456, hereby the file should be named as 123456.zip ), and then submit the .zip file via BB system.

Please note that, the teaching assistant may ask you to explain the meaning of your program, to ensure that the codes are indeed written by yourself. Please also note that we may check whether your program is **too similar** to your fellow students' code using BB.