INPUT	OUTPUT
December 21, 2012 ? (Gregorian)	5773 Teveth 8 (Hebrew) 1434 Safar 7 (Islamic) 1934 Agrahayana 30 (Indian Civil) 13.0.0.0 (Mayan Long Count)

14.8 Calendrical Calculations

Input description: A particular calendar date d, specified by month, day, and year.

Problem description: Which day of the week did d fall on according to the given calendar system?

Discussion: Many business applications need to perform calendrical calculations. Perhaps we want to display a calendar of a specified month and year. Maybe we need to compute what day of the week or year some event occurs, as in figuring out the date on which a 180-day futures contract comes due. The importance of correct calendrical calculations was perhaps best revealed by the furor over the "Millennium bug"—the year 2000 crisis in legacy programs that allocated only two digits for storing the year.

More complicated questions arise in international applications, because different nations and ethnic groups use different calendar systems. Some of these, like the Gregorian calendar used in most of the world, are based on the Sun, while others, like the Hebrew calendar, are lunar calendars. How would you tell today's date according to the Chinese or Islamic calendar?

Calendrical calculations differ from other problems in this book because calendars are historical objects, not mathematical ones. The algorithmic issues revolve around specifying the rules of the calendrical system and implementing them correctly, rather than designing efficient shortcuts for the computation.

The basic approach underlying calendar systems is to start with a particular reference date (called the *epoch*) and count up from there. The particular rules for wrapping the count around into months and years is what distinguishes one system from another. Implementing a calendar requires two functions, one that, given a date, returns the integer number of days that have elapsed since the epoch, the other of which takes an integer n and returns the calendar date exactly n days

from epoch. These are exactly analogous to the ranking and unranking rules for combinatorial objects such as permutations (see Section 14.4 (page 448)).

That the solar year is not an integer number of days long is the major source of complications in calendar systems. To keep a calendar's annual dates in sync with the seasons, leap days must be added at both regular and irregular intervals. Since a solar year is 365 days and 5:49:12 hours long, adding a leap day every four years leaves an extra 10 minutes and 48 seconds unaccounted for each year.

The original Julian calendar (from Julius Caesar) ignored these extra minutes, which had accumulated to ten days by 1582. Pope Gregory XIII then proposed the Gregorian calendar used today, by deleting the ten days and eliminating leap days in years that are multiples of 100 but not 400. Supposedly, riots ensued because the masses feared their lives were being shortened by ten days. Outside the Catholic church, resistance to change slowed the reforms. The deletion of days did not occur in England and America until September 1752, and not until 1927 in Turkey.

The rules for most calendrical systems are sufficiently complicated and pointless that you should lift code from a reliable place rather than attempt to write your own. We identify suitable implementations next.

There are a variety of "impress your friends" algorithms that enable you to compute in your head what day of the week a particular date occurred. Such algorithms often fail to work reliably outside the given century and certainly should be avoided for computer implementation.

Implementations: Readily available calendar libraries exist in both C++ and Java. The Boost time-data library provides a reliable implementation of the Gregorian calendar in C++. See *http://www.boost.org/doc/html/date_time.html*. The Calendar class in `java.util.Calendar` implements the Gregorian calendar in Java. Either of these will likely suffice for most applications.

Dershowitz and Reingold provide a uniform algorithmic presentation [RD01] for a variety of different calendar systems, including the Gregorian, ISO, Chinese, Hindu, Islamic, and Hebrew calendars, as well as other calendars of historical interest. *Calendrical* is an implementation of these calendars in Common Lisp, Java, and Mathematica, with routines to convert dates between calendars, day of the week computations, and the determination of secular and religious holidays. *Calendrical* is likely to be the most comprehensive and reliable calendrical routines available. See their website at *http://calendarists.com*.

C and Java implementations of international calendars of unknown reliability are readily available at SourceForge (*http://sourceforge.net*). Search for "Gregorian calendar" to avoid the mass of datebook implementations.

Notes: A comprehensive discussion of calendrical computation algorithms appear in the papers of Dershowitz and Reingold [DR90, RDC93], which have been superseded by their book [RD01] that outlines algorithms for no less than 25 international and historical calendars Three hundred years of calendars representing tabulations for all dates from 1900 to 2200 appear in [DR02].

Concern exists in certain quarters whether December 21, 2012 represents the end of the world. The argument rests on it being the date the Mayan calendar spins over to 13.0.0.0.0 after a 5,125 year cycle. The reader should rest assured, since I would never have devoted so much effort to writing this book were the world to be ending so imminently. The Mayan calendar is authoritatively described in [RD01].

Related Problems: Arbitrary-precision arithmetic (see page 423), generating permutations (see page 448).