Project 7

due at midnight on Mon Nov 16 (60 points)

In this assignment, we will use a loop to print a simple calendar. It begins by asking the user for the year and month, in numeric form, and then it prints the calendar with a nice title, like this:

```
Enter year (1900-): 2013
Enter month (1-12): 11

    ** November 2013 **

 Sun Mon Tue Wed Thu Fri Sat
                       1   2
   3   4   5   6   7   8   9
  10  11  12  13  14  15  16
  17  18  19  20  21  22  23
  24  25  26  27  28  29  30
```

Call your program p7cal.cpp and submit to this dropbox for project 7.

Error checking

You should check the inputs of year and month for errors. (Year should be 1900 or later, and month should be from 1 to 12.) This time, though, use a loop so that after an error the user has another chance to enter the number. Here's a transcript of the desired loop behavior:

```
Enter year (1900-): 1492
ERROR. Enter year (1900-): 1580
ERROR. Enter year (1900-): -2013
ERROR. Enter year (1900-): 1972
[Given a correct input, the program moves on...]
Enter month (1-12): ^C
```

Determining start day

One of the hard problems of printing a calendar is figuring out on which day it starts. There are a variety of algorithms for that, but here is the simplest. It uses an **array,** which we haven't learned yet – but you don't really need to understand the details to use it:

```
static int t[] = {0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};
int y = year - (month < 3);
int dayOfWeek = (y + y/4 - y/100 + y/400 + t[month-1] + 1) % 7;
```

If you ensure your input variables `year` and `month` have valid values before reaching the code above, then this code will ensure that `dayOfWeek` indicates on which day that month starts. It uses `0` to represent Sunday, `1` for Monday, up through `6` for Saturday.

You may want to test it by printing out `dayOfWeek` and then enter different years and months. Verify them against a calendar online.

Producing the calendar grid

There are several ways to approach this. My favorite is first to just print all the integers from 1 to 31. (Later you can substitute 30 or 29 based on the month… don't worry about leap years.) You can find a loop in the notes that prints increasing numbers.

Now you'll have something like this:

```
 Sun Mon Tue Wed Thu Fri Sat
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

and you need to think about two problems:

1. Shifting the `1` over far enough that it lands on the right day. (You would do this by inserting some number of spaces, depending on `dayOfWeek`)
2. Inserting newlines after any day that lands on Saturday.

One way of thinking about the second problem is using modular arithmetic. For months that start on Friday (`dayOfWeek==5`), you break on the 2nd, 9th, 16th, 23rd, etc. What these numbers have in common is that `day%7 == 2`.

Now think about a month that starts on Wednesday (`dayOfWeek==3`). In that case, you break on 4th, 11th, 18th, 25th, etc. and for these `day%7 == 4`.

The pattern is that the modulus that indicates a Saturday is always `(7 - dayOfWeek) % 7`.