EEN118 LAB FIVE

In this lab you will use clean structured design to solve a problem that is normally considered to be very difficult, and find that it is in fact surprisingly easy. *Look before you leap*: think about how your program is going to be organised, don't just start typing. A rational design will give you a working program quite easily; an unplanned design will not. Read the whole lab before starting the first part. This week, you may use either unix or windows for development.

You will be creating a nicely formatted calendar. For everything except the extra credit part, you only need to make it work correctly for the years 1901 to 2099. This restriction simplifies things somewhat.

1. *Length of a Month*

   Design a function that takes two parameters: year and month, and returns an integer indicating how many days long that month is. January 2003 is 31 days long, February 2003 is 28 days long, February 2004 is 29 days long, and so on. Remember that for February, leap years must be taken into account. Between the years 1901 and 2099 the leap-year rule is simple: a year is a leap year if it is divisible by 4.

   Incorporate your function into a simple program that allows you to test it conveniently, and then test it. Each stage of this lab assignment depends on the previous stage, so you won't do any good by going ahead with an incorrect function.

2. *Day of the Year*

   Design a function that takes three parameters: year, month, and day, and returns an integer indicating which day of the year that date is. For example, the 1st of January is day 1 of the year, the 2nd of January is day 2, the 1st of February is day 32, and so on.

   Incorporate your function into a simple program that allows you to test it conveniently, and then test it thoroughly.

3. *Day of the Millennium*

   Design a function that takes three parameters: year, month, and day, and returns an integer indicating which day of the millennium that date is. Forget foolish arguments about whether the millennium started in 2000 or 2001; base your calculations on 2000 and everything will work out nicely. For example, the 1st of January 2000 was day 1 of the millennium; the 31st of December 2000 was day 366, the 1st of January 2001 was day 367, and so on. You only need to make this part work for years 2000 to 2099.

   Incorporate your function into a simple program that allows you to test it conveniently, and then test it thoroughly.

4. *Extending the Range*

   Extend your solution to part 3 so that it works for all years between 1901 and 2099. The rule for deciding leap years does not change, all you have to do is make sure that negative dates are handled correctly.

   As 1/1/2000 is day 1, the day before it, 31/12/1999 must be day 0 (not day -1, because then there would be one missing), 30/12/1999 must be day -1, and so on: 1/1/1999 is day -364, 31/12/1998 is day -365.

   If you have trouble with this part, ask the Lab Guy for assistance, although it is possible to continue to part 5 without this part working, it would be a shame.

5. *Day of the Week*

   Extend your solution to part 4 to produce a function that takes a date as before with the three components year, month, and day, and returns a number in the range 0 to 6 indicating the day of the week, 0=Sunday, 1=Monday, 2=Tuesday, ..., 6=Saturday. For example, the 27th of February 2003 is a Thursday, so `DayOfWeek(2003, 2, 27)` should return `4`. There is a very very simple way to do this. Think about it for a while, but if you're stuck ask the Lab Guy for a clue.

6. *A Calendar for a Month*

   Use your solution to part 5 in a program that allows the user to enter two integers, representing a year and month, and displays a correctly formatted calendar for that particular month. You may use either plain text output (with the `print` function or `cout`) or graphics (with the `WriteText` function). Align your weeks in the traditional manner so that Sunday appears first, and Saturday last.

   Things to consider: how many spaces should be printed before the number 1? (or how many pixels should be skipped)? This is just a multiple of the day-of-week number for the first of the month. When should you do a newline? (or when should you set `x` back to zero and increase `y`)? This is simply whenever you have just printed a Saturday. How do you keep the columns properly aligned? That one's easy.

   This is the sort of thing we should see for an input of 2006 9:

```
Su Mo Tu We Th Fr Sa
                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
```

7. *Pretty*

   Make it print the month and year nicely cented above the first line: "`September 2006`".

8. *Extra Credit*

   Assuming you used plain text output to get started, now make a nice graphical calendar, with well-chosen fonts and everything. Make your calendar program work for ANY year, not just restricted to the range 1901 to 2099. The complete leap year rules are given on the next page. Read the note at the end of page 3 for details of the requirement. Perhaps even make it draw a whole year's calendar.

For reference:

Rules for Leap Years under the Gregorian Calendar

A year is a leap year if it is divisible by 400, and
A year is a leap year if it is divisible by 4 but not by 100.
Those are the only rules.

So the years...

> 2000, 2400, 2800, 3200 are leap years
> 1800, 1900, 2100, 2200, 2300, 2500 are not leap years
> 1904, 1908, 2004, 2008, 1996, 2096, 2388 are leap years
> 1901, 1905, 2001, 2003, 2006, 2998, 3007 are not leap years.

The Gregorian calendar was only introduced in the English speaking world on 14th September 1752. Before then, the Julian calendar was in use.

Under the Julian calendar, the leap year rule was much simpler: a year is a leap year if it is divisible by 4. That's it.

The changeover period was somewhat confusing. So you can get the idea, here are the calendars for August, September, and October of 1752, as you can see, the 3rd to 13th of September just didn't happen that year:

```
     August 1752            September 1752            October 1752
Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
                   1           1  2 14 15 16     1  2  3  4  5  6  7
 2  3  4  5  6  7  8    17 18 19 20 21 22 23     8  9 10 11 12 13 14
 9 10 11 12 13 14 15    24 25 26 27 28 29 30    15 16 17 18 19 20 21
16 17 18 19 20 21 22                            22 23 24 25 26 27 28
23 24 25 26 27 28 29                            29 30 31
30 31
```

You do not need to make you program work for the Julian calendar for the extra credit part; you can limit it to dates after September 1752 without penalty. But for *extra* extra credit, you can try to go the whole way.