CSC 180 Assignment 1, Fall 2001

Due at the end of Thursday October 4, 2001; no late assignments without written explanation.

Calculations involving dates and times are often more complex than one might think. There are a number of interesting programming problems in this area.

This assignment involves the calculation of the day of the week (e.g. "Sunday") from the year, month, and day number of a given date. There is a standard algorithm for this, as follows.

- Let D be the day number, from 1 to 31. E.g. for Sept 3, 2001, D=3.
- Let M be a special month number counting from March as follows: For January, M=11; for February, M=12; for March, M=1; and so on to December for which M=10.
- If it's January or February, subtract one from the year.
- Let C be the century portion of the adjusted year (e.g. for 1937, C=19).
- Let Y be the last two digits of the adjusted year, i.e. the year without the century (e.g. for 1937, C=37 . . . or 36 if the month is January or February).
- Compute a value as follows, where "div" is integer division (division without remainder or fractions): (26M − 2) div 10 + D + Y + Y div 4 + C div 4 − 2C + 7777
- The remainder when this value is divided by 7 (that is, this value mod 7) is the weekday number, where 0 is Sunday, 1 is Monday, and so on through 6 which is Saturday.

A weekday-calculating function

Write a function "weekday", declared as "`int weekday(int year, int month, int day)`", which takes a date as three integers and returns the weekday number using the above algorithm. It does not print anything; it returns a value, like the math library functions, for the caller to do what they want with it.

Compiled with the following function in a separate file:

```
#include <stdio.h>

int main()
{
    extern int weekday(int year, int month, int day);
    printf("%d\n", weekday(1937, 6, 18));
    return 0;
}
```

it should output 5, because June 18th, 1937, was a Friday. The above file can be copied from ~ajr/a1/testweekday.c on an ECF machine.

A short main program to call weekday()

Write a main function in a separate file, main.c, which prompts for the year, month, and day of a date, and calls the weekday() function you wrote above. If you call scanf() with a format argument of "%d%d%d", it will return 3 if the user enters the three integers properly. Otherwise, say "Bad input" (a better message would be possible, but is not the focus of this assignment) and

(over)

do not proceed.

The user's input should then be checked. Any year is acceptable, including negative years, but the day number must be between 1 and 31 (inclusive) and the month number must be between 1 and 12 (inclusive). If they enter an invalid day or month number, say "Invalid date" and do not proceed.

Otherwise, call the weekday function, and output the answer in the following format. The entire interaction looks like this, where "1937 6 18" is the user input.

```
Input year, month, and day, in that order
1937 6 18
The 18th day of the 6th month in the year 1937 is weekday number 5
```

To assist our use of the grading software, please format your output exactly as above, and use those exact two error messages ("Bad input" and "Invalid date").

A main program with no extraneous output

Make a version of that main.c in a new file, toolmain.c, which is less "user-friendly" but perhaps more useful as a tool in conjunction with other programs or scripts. It will not output a prompt, and it will not label the output. It immediately does a scanf; it still checks for errors; but if there are no errors it simply outputs the weekday number with no labelling.

The entire interaction would look like this, where "1937 6 18" is the user input.

```
1937 6 18
5
```

The first Monday after a given date

Write a program which uses weekday() to find the first Monday starting from a given date. For example, June 14th, 1937, is a Monday. If the user enters that date ("1937 6 14"), it is simply output. If the user enters the date "1937 6 18", which is not a Monday, your program increments the day number in a loop until it arrives at 1937/06/21, which is the next Monday.

To avoid repeating the code from above, this program does not need to check the return value from scanf nor check that the month and day numbers are in the correct range. It can just call scanf() and ignore the return value, and proceed with the calculation.

This program should not prompt for the input, either.

However, its output should be nicely formatted in accordance with the following example, where again, "1937 6 18" is the user input.

```
1937 6 18
1937/06/21 is a Monday
```

Optionally, you can use the '0' printf modifier to print leading zeroes, and a field width to specify how wide you want the field to be. The above "06" and "21" were formatted with "%02d" instead of the simple "%d".

You do not need to deal with the case where the next Monday is in the next month.

(continued)

File submission

Each file must begin with a "prologue comment" including any useful description of the file and also your full name, your student number, and your tutorial room and time (on Tuesday—the tutorial, not the lab session).

Your program should use proper indentation as illustrated by programs in lecture and in the textbook.

Your files ***must*** have the names weekday.c, main.c, toolmain.c, and firstmonday.c, and they must obey the inter-file interface specifications above. That is, your weekday.c must work with the example main function from page 1 in a separate file, and all of your variant main functions in main.c, toolmain.c, and firstmonday.c must work with your weekday.c (or anyone else's correct weekday.c, such as mine).

When you are done, submit your files for grading. You submit the source code files, not the compiled files. Submit your files with the command

```
submitcsc180f 1 weekday.c main.c toolmain.c firstmonday.c
```

You may still change your files and resubmit them with the same command any time up to the due time. You can check that your assignment has been submitted with the command

```
submitcsc180f -l 1
```

(Note that the option is "–l" with the letter L, for list, although the argument at the end is "1", the digit, for assignment 1.) This is the only submission method; you do not turn in any paper.

Remember: This assignment is due at the end of Thursday, October 4th at midnight *sharp*. Late assignments are not ordinarily accepted and *always* require a written explanation. Even if you are just a few minutes late, you must submit a written explanation for lateness or we will not know to check back for your assignment. Check the submission time with "submitcsc180f –l 1".

If you are not finished your assignment by the submission deadline, you should just submit what you have, for partial marks. Getting two out of the 5% isn't *quite* the end of the world if you just do it once (although obviously it's suboptimal), and it's certainly better than a zero. Most of all, I don't want to create the situation where you get further and further behind in all your work and lose more and more marks. There's a time allotted to each assignment; you do what you can in the time allotted.

Despite the above, I'd like it to be clear that if there *is* a legitimate reason for lateness, please do submit your assignment late and give me that written explanation. If you know in advance that you're going to be late for a good reason, please talk with me about it in advance; although be warned that my suggestion might be that you should simply submit in advance.

I'd also like to point out that even a zero out of 5% is much better than cheating and suffering an academic penalty. Don't cheat even if you're under pressure. Whatever the penalty eventually applied for cheating, it will be worse than merely a zero on the assignment.