Assignment

Date reading & date difference
(due before 4pm on Monday of week 8)

Dr. Oscar Mendez

Outline

The first assignment will be to read and sort a list of dates.
You will build up the program in three stages.

- **Part I**: date reading and difference
- **Part II**: Leap years
- **Part III:** Sorting

Note that in each part below, any output from your program should go to **stdout** (using **printf** statements) and NOT stderr.

Part I

- Read 2 dates
- Determine the number of days between them
- Dates can be entered in two formats:

  - **<day>/<month>/<year>**
  - **<day>-<month>-<year>**
  - (where <day>, <month> and <year> are integers)

- **(for now you should ignore leap-years)**

Part I (cont'd)

- Create your program in a file **date1.c**
- Program must
  - Read the two dates from the standard input
  - Output the difference to the standard output
  - If a date is entered incorrectly or invalid (year in [1, 10000], month between [1-12], day depend on month)
    - Error to standard error (NOT output)
    - Date must be entered again
- Compile using:

```
$ gcc -Wall -ansi date1.c -o date1
```

Part I (cont'd)

For example, to print the difference between the two dates 23/11/1987 and 5/11/1987 you would type:

```
./date1
```

to start the program running, and then type:

```
23/11/1987
5-11-1987
```

hitting the return key after each line to send the dates to the program, The program should then print out the answer:

```
-18
```

indicating that the second date is 18 days before the first date.

Part II

- Start by copying your date1.c into a new file date2.c

- Modify the new program to handle leap years, i.e. allow for an extra day in February in years which are a multiple of four (forget about the other rules for century, etc.).

Part III

- Copy your date2.c to a new file date3.c
- Modify the program to read more than 2 dates.
- When the program runs, the first thing a user types should be the number of dates. Each date is then read one after another. To input 3 dates:

```
3
1/3/1998
3/2/1956
19/11/1901
```

- **HINT: if you write functions, you can call them as many times as needed.**

Part III (cont.)

- After reading as many dates as specified, the program should then sort them into ascending order, and display them back to the user.

```
19/11/1901
3/2/1956
1/3/1998
```

- NOTE: as in part 1, **invalid date entries should be ignored**. The program should only count valid dates.

Warning!

- The program must work ***exactly*** as specified
  - No other printout or message to stdout (you are free to write other messages to stderr)
  - Invalid dates or format must be detected & generate the error message to stderr
- Good style is expected
  - Your code must be commented
  - Use functions, avoid globals etc.
- Make sure the code compiles and work on university (linux) PCs
- … and test, test, test...

Testing

- One example is provided for each part, to help test your code
- You can test in the same way as the automated system using:

```
$ ./date1 < examples/part1/in > tmp
$ diff -s tmp examples/part1/out
```

Submission

BEFORE starting, read the rules and the submission procedures on Surreylearn.

Submit using Surreylearn.

Submission deadline: **4pm Monday of Week 8** (strict!)

Submission (cont'd)

The name of the submitted file MUST be preceded with your surname and initial followed by the name of the program.

For example, if your surname is ``Brown'', and your first name is ``Peter'', you need to name the file as follows:

BROWNP-date1.c

Plagiarism

- If two (or more) assignments appear to have the same origin, **none** of them will count.

- So, to be safe, protect your work directory (see lab 1), using

```
$ chmod -R 700 ./clabs/
```

Have fun!