CS253 HW3: Options!


Tibetan calendar and astrological diagram

Description

For this assignment, you will improve upon your previous work in HW1, adding command-line options, and reading from files *or* standard input.

Arguments

The first command-line arguments should be *options*:

`-f` *format*
: Specify the *format*, to be given to `strftime()`. For example, a format of **`"%A %B %d"`** would give output of **`Monday May 11`** for a date of today.

`-i`
: Specify a format that will write the date in the ISO 8601 format of YYYY-MM-DD, exactly four digits (year), a hyphen, exactly two digits (month), another hyphen, and exactly two more digits (day of month). Today would produce **`2020-05-11`**.

`-v`
: Announce, to standard output, each file as it is read. Display **`*** Processing`** *filename* for each file, and **`*** Processing standard input`** for standard input.

Any remaining arguments are files that should contain one date per line. If no files are given, then read from standard input.

Input Format

Input lines can be in any of these formats:

year`.`*day*
: As in HW1. The *year* should be 1≤*year*≤9999, and the *day* is either 1≤*day*≤365 or 1≤*day*≤366, depending on leap year. Today is 2020.132.

YYYY-MM-DD
: YYYY is a four digit year. MM is a two-digit month. DD is a two digit day. Today is 2020-05-11. **`YYYY`** should be 1≤**`YYYY`**≤9999, **`MM`** should be 1≤**`MM`**≤12, and **`DD`** should 1 to however many days are in that month that year.

today
: The current day. Literally, the five characters "today", in either case. E.g., "tOdAy" is valid.

yesterday
: The day before today. Literally, the nine characters "yesterday", in either case. E.g., "YEstERday" is valid.

tomorrow
: The day after today. Literally, the eight characters "tomorrow", in either case. E.g., "TomORRow" is valid.

Output Format

For each input line, translate it using the given format, and write it to standard output, followed by a newline.

Sample Runs

Here are sample runs, where `%` is my prompt.

```
% cat pearl
1941-12-07
00000000001941.00000341
% cat limits
1.1
0001-01-01
9999.365
9999-12-31
% echo ToDay | ./hw3
Wed Feb 12 2020
% echo YEStErDaY | ./hw3 -f 'Day %d of the month of %B of the year %Y'
Day 11 of the month of February of the year 2020
% echo 2021.1 | ./hw3 -f 'Week-based year: %G%nConventional year: %Y'
Week-based year: 2020
Conventional year: 2021
% ./hw3 -f"A day that will live in infamy: %A %B %e %Y" -v pearl
*** Processing pearl
A day that will live in infamy: Sunday December  7 1941
A day that will live in infamy: Sunday December  7 1941
% ./hw3 -f"A day that will live in infamy: %A %B %e %Y" -v <pearl
*** Processing standard input
A day that will live in infamy: Sunday December  7 1941
A day that will live in infamy: Sunday December  7 1941
% ./hw3 -vi limits pearl
*** Processing limits
0001-01-01
0001-01-01
9999-12-31
9999-12-31
*** Processing pearl
1941-12-07
1941-12-07
% cat pearl | ./hw3 -f '%04Y %b %d %a' -v limits
*** Processing limits
0001 Jan 01 Mon
0001 Jan 01 Mon
9999 Dec 31 Fri
9999 Dec 31 Fri
```

Yes, that last example really is correct. Only read standard input if no filenames are given.

Requirements

- Error messages:
  - go to standard error
  - include the program name, no matter how it was compiled
  - include the entire input line, if not of one of the acceptable formats, or has an invalid value.
- Produce an error message and stop the program if:
  - an option is bad
  - a file cannot be opened
  - an input line is not of one of the acceptable formats.
- If more than one problem exists, you don't have to report them all. Produce one error message and stop.

- Options:
  - An invalid **`-f`** *format* string (e.g., **`-f %Q`**) has undefined results.
  - A format string with hour/minute/second specifiers (e.g., **`%H`**, **`%M`**, **`%S`**, **`%c`**) has undefined results.
  - You may assume that the **`-f`** *format* string will result in no more than 64 characters.
  - It is an error to specify **`-f`** more than once, or to specify both **`-f`** and **`-i`**.
  - If neither **`-f`** nor **`-i`** is given, use the format shown in the examples, which is *different* than that of HW1.
  - The **`-i`** and **`-v`** options may be specified multiple times, with no additional effect.
  - Options must precede filenames. **`./hw3 -i infile -v`** must attempt to process the file **`-v`**, which would probably fail.
  - Bundling of options must work:
    - **`./hw3 -vi data1 data12653932`** is the same as **`./hw3 -v -i data1 data12653932`**
    - **`./hw3 -f'%a/%b' -v data`** is the same as **`./hw3 -f '%a/%b' -v data`**
    - **`./hw3 -vf "date: %c"`** is the same as **`./hw3 -v -f "date: %c"`**
    - **`./hw3 -fv "%c"`** is the same as **`./hw3 -f v "%c"`** (which will treat **`v`** as a format, and **`%c`** as a filename)
    - **`./hw3 -v-f "%c" data`** is invalid.

- The output must end with a newline.
  - Newlines do not separate lines—newlines *terminate* lines.
- Creativity is a wonderful thing, but your output format is *not* the place for it. Your non-error output should look exactly like the output shown above. You have more leeway in error cases.
  - UPPERCASE/lowercase matters.
  - Spaces matter.
  - Blank lines matter.
  - Extra output matters.

- You may not use any external programs via `system()`, `fork()`, `popen()`, `execl()`, `execvp()`, etc.
- You may not use C-style I/O facilities such as `printf()`, `scanf()`, `fopen()`, and `getchar()`.
  - Instead, use C++ facilities such as `cout`, `cerr`, and `ifstream`.
- You may not use dynamic memory via `new`, `delete`, `malloc()`, `calloc()`, `realloc()`, `free()`, `strdup()`, etc.
  - It's ok to *implicitly* use dynamic memory via containers such as `string` or `vector`.
- You may not use the `istream::eof()` method, even if called via other syntax such as **`cin.eof()`**.
- No global variables.
  - Except for an optional single global string containing `argv[0]`.
- For readability, don't use ASCII `int` constants (**`65`**) instead of `char` constants (**`'A'`**) for printable characters.

- We will compile your program like this: **`cmake . && make`**
  - If that generates warnings, you will lose a point.
  - If that generates errors, you will lose *all* points.
- There is no automated testing/pre-grading/re-grading.
  - Test your code yourself. It's your job.
  - Even if you only change it a little bit.
  - Even if all you do is add a comment.

If you have any questions about the requirements, **`ask`**. In the real world, your programming tasks will almost always be vague and incompletely specified. Same here.

Hints

- Use `getopt`. Seriously, use it. Don't do this yourself. You will have to actually *read* the manual page for `getopt`. This is a skill that you must acquire to be a good programmer.

Tar file

- For each assignment this semester, you will create a tar file, and turn it in.
- The tar file for this assignment must be called: **`hw3.tar`**
- It must contain:
  - source files (**`*.cc`**)
  - header files (**`*.h`**) (if any)
  - **`CMakeLists.txt`**
- These commands must produce the program **`hw3`** (note the dot):

```
cmake . && make
```

- *At least* **`-Wall`** must be used every time `g++` runs.

How to submit your homework:

- Use web checkin, or Linux checkin:

```
~cs253/bin/checkin HW3 hw3.tar
```

How to receive *negative* points:

Turn in someone else's work.