$Id: asg1j-jcal-3darray.mm,v 1.36 2015-01-02 18:32:56-08 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Assignments/asg1j-jcal-3darray

1. Overview

Write a program which displays the calendar for any particular month or year. Its output should be the same as the `cal`(1) utility. Objectives : Unix commands and the command line, introduction to Java and its libraries, `gmake`.

Before your begin, experiment with `cal` to see what its output looks like. Your program will be similar, but with different options. Note that a single operand is a year and two operands consist of a month followed by a year. Also note that `cal 9 69` refers not to the month when Unics was created, but rather to *mensis September DCCCXXII a.u.c.*.

Your program is to use the class name `jcal`, reside in the file `jcal.java`, make the class file `jcal.class`, and finally be placed in the executable file `jcal`. Your synopsis will be as follows, with none of the options listed in the Linux man page :

SYNOPSIS
`jcal` [[*month*] *year*]

2. Implementation Strategy

How do you go about writing a program that is larger than trivial ? It is necessary to begin with a small piece and add to it, a little piece at a time. In the end, the entire program will then be written. Following is a sequence of steps in constructing this program :

(1) The subdirectory `misc/` contains sample Java programs for you to study before beginning your program. The subdirectory `code/` contains the files you should start with for your program.

(2) As you are programming, you should always be ready to refer to the reference manual. For Java, that is at
`http://docs.oracle.com/javase/7/docs/api/`
The specific packages you will need are `java.lang` and `java.io`. In addition, for this assignment, `java.util.GregorianCalendar` and its ancestor `java.util.Calendar`.

(3) Keep in mind that code can get quite complex, and to keep it simpler, make sure that no function exceeds about 20–30 lines of code. Whenever a function gets too large, break it up into several functions. Any time you have duplicate code, extract the duplicated code into a new function and call that new function from where the duplicate code used to be.

(4) Look over the sample program `misc/monthcal.java`, which shows how to use Java's `GregorianCalendar` class to iterate over a month and determine the day of the week for each day of the month. It also checks for the end of the month, which occurs when adding 1 to the day causes the month to change and the day to revert to 1. Run it with various command line arguments.

(5) Make sure your program does not crash on a `NumberFormatException` when given bad arguments. Look at `misc/integerarg.java` to see how to catch the error. Of course, you should be printing a usage message to the standard error

in the same format as `cal`(1).

(6) Extract some code from that example and write a method `make_month` that will accept a month and a year as arguments and return some data structure representing a month that is suitable for easy printing. This method should not do any printing. The purpose of this function is to separate the complexity of calculating the month from its printing, since you will have to print months side by side in year format. Start with:

```
int[][] make_month (int month, int year) {
   int[][] month = new int[WEEKS_IN_MONTH][DAYS_IN_WEEK];
   ...
}
```

(7) A good data structure for a month is `int[][]` and for a year, `int[][][]`. A year is an array of 12 months. A month is an array of 6 weeks. A week is an array of 7 days. A day is an integer. Look at the full year for 2015 in Figure1 (page 4), and at March, 2015, in Figure 2 to see why you need 6 weeks. For days not within a month, you can use 0 as the array element.

(8) Your main program should just print out the result of this then, and do it only for the current month. This step is just a testing phase which will be removed later. Call this function `print_month`.

(9) Add in a method which will analyze the command line arguments and return a month and a year or print an error message if the arguments are not good ones. Your program should work exactly like `cal`. Output always goes to the standard output, but errors go the standard error, not the standard output. See `System.err` and `System.exit`. Print error messages with `misclib.warn` or `misclib.die`.

(10) To verify that your exit status codes are correct, use the command

```
echo $?
```

In Java, the function `System.exit` is used to stop the program and return an exit code. Returning from the `main` function is equivalent to `System.exit(0)`.

(11) The program must have zero, one, or two command line arguments. Verify that the month, if present, is in the range 1–12, and that the year, if present, is in the range 1–9999. Print a message and exit if not.

(12) Now add in a function to compute calendars for the entire year. This is just a matter of `make_year` calling `make_month` 12 times.

(13) Write a function `print_year` to print out a year in the year format, and a function `print_month` to print a single month when the calendar requires that. Try to avoid duplicating any code.

(14) Try various arguments and calling sequences, both valid and invalid. Check the output from each. Are the return codes the same as for `cal`(1)? What about the error messages to standard error and the output to standard output? You can use the command `diff`(1) to check the output of your program against the standard Utility. What about September, 1752? What about years before 1 and after 9999? What about silly months? What about words instead of numbers?

(15) For each of the tests that you run, compare your output to that produced by **`cal`**. This can be done with shell commands such as :

```
cal 2015 >cal.2015.out 2>cal.2015.err
jcal 2015 >jcal.2015.out 2>jcal.2015.err
diff cal.2015.out jcal.2015.out
```

The **`diff`**(1) command should produce no output. If it does, then the programs differ. Your program should produce identical output to that of **`cal`**.

```
bash-3$ cal 2015
                               2015

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

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


bash-4$
```

Figure 1. Output of `cal` for a year

```
bash-5$ cal 9 1752
   September 1752
Su Mo Tu We Th Fr Sa
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30



bash-6$
```

```
bash-7$ cal 1 2015
    January 2015
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 31

bash-8$
```

Figure 2. A short month and a long month

3. The Julian and Gregorian Calendars

According to legend, the Roman calendar, was instituted by Romulus in about 753 BCE, but actually probably deriving from the Greek lunar calendar. That Roman calendar had 10 months (304 days) per year, Mārtius, Aprīlis, Maius, Iūnius, Quīnctīlis, Sextīlis, September, Octōber, Nouember, and December. plus about 61 days during the winter which were not counted. King Numa Pompilius, about 713 BCE, added Iānuārius and Februārius, making the year 355 days long, with an intercalary month, Mercedonius, every so often. In 45 BCE, Gāius Iūlius Caesar reformed the calendar to the 365 days we know, plus an extra day every fourth year, and renamed Quīnctīlis to Iūlius, after himself. Imperātor Gāius Iūlius Caesar Octāuiānus Augustus renamed Sextīlis to Augustus. Tiberius refused to have September named after himself, asking, "What would happen after Rome has had more then 12 emperors ?"

The Gregorian calendar was introduced in Italy on October 15, MDLXXXII. Astronomer Christopher Glavius actually did the work, but Pope Gregory XIII put his name on it. On September 14, MDCCLII, King George II adopted the Gregorian calendar for the British Empire. Prior to that date, the Julian calendar was used. So, for the British Empire and her colonies, September, 1752, is a rather strange month, as shown in Figure 2 (page 5).

The Gregorian calendar has a leap year every 4 years, except that years divisible by 100 are not leap years, but years divisible by 400 are. Hence 2000 was a leap year, but 1900 was not and 2100 will not be. The average length of a Gregorian calendar year is thus 365.2425 days, whereas the actual length of the tropical year 2000 was 365.242190 days. The Gregorian calendar is thus inaccurate by one day in 3225.8 years (26.784 seconds per year). By contrast, the Julian calendar, with a leap year every 4 years, was inaccurate by one day in 128.0 years (674.784 seconds per year = 11.2464 minutes per year). The current day length is about 86400.003 seconds. And a second is defined such that ^{133}Cs has an atomic frequency of 9192631770 Hz.

4. Internationalization (a.k.a. I18N) of the Base Assignment

The base level assignment should work in exactly the way that **cal**(1) does, including output to `stdout` and `stderr`, and it should return the same exit status codes. But it does not handle `cal`'s numerous options. This part adds additional functionality by allowing output in multiple different languages. (The reason the abbreviation is I18N is that there are 18 letters between the "i" and the "n" in the word "internationalization".) First study the example program `misc/i18nmonth.java`, and import the package `java.util.Locale` into your program. These packages, used as in the example program, can be used to translate the month and day names into the appropriate language. Your new synopsis will be :

SYNOPSIS

`jcal` [–*locale*] [[*month*] *year*]

(1) The *locale* option will be specified as any one of the ISO-639 language names supported by Java. The sample program `misc/localeinfo.java` prints them all out, but is not part of the assignment. It just shows you what you might use as the locale argument. So, for example, with various options, output can be in Dansk (`-da`), English (`-en`), español (`-es`), français (`-fr`), Gaeilge (`-ga`), italiano (`-it`), Nederlands (`-nl`), norsk (`-no`), português (`-pt`), suomi (`-fi`), svenska (`-sv`), íslenska (`-is`), etc. Of course, since our `xterm`(1) programs use ISO-8859-1 (ISO-Latin-1) which supports the langauges of Western Europe, those languages that use non-Latin characters will not appear properly. Unicode supports them all.

(2) Print out all of the day names as the first two letters of the name of the day, and the month names either as the first three letters or as the complete name in the same way that `cal`(1) does. If no locale is specified your output must be identical to that of `cal`. The program `misc/printcalnames.java` shows you how to obtain the names of days and months in foreign languages. You will only be using the single-argument form of `new Locale`.

(3) Another program, `misc/i18nmonth.java` does much the same and is another example. Note that for languages which do not use the Roman alphabet, the output is gibberish. If you have a terminal capable of handling other languages, you might try Arabic (`-ar`), Ἐλληνικά (`-el`), Hebrew (`-iw`), Japanese (`-ja`), Russian (`-ru`), Chinese (`-zh`), etc., but obviously the grader will not do any of that.

(4) The Julian to Gregorian changeover date varies from country to country, and you may do a web search to find out a list of days for each country. However, since you must emulate `cal`(1), you should hard code the British Empire's changeover date of September 14, 1752.

5. What to Submit

Look in the subdirectory `.score` for instructions to the graders. The file `SCORE` contains detailed instructions. The scripts `mk.build` and `mk.test` will be used to test your program. You should use these scripts before the graders see your code. That way you can attempt to grade your program yourself.

Submit early. Submit often. See lab 1 for more information on how to verify your submit. If you submit the wrong version of your code or forget to submit a file, you will lose many points. Submit the following files : `jcal.java` and `misclib.java`, containing your source code ; `Makefile`, which will build the jar file `jcal` from the source file, with the target `all` being first and which builds `jcal`; `README`, which should contain your name and username and any specific things the grader should know. Did you do the internationalized version?

If you are doing pair programming, carefully read the document in `cmps012b-wm/Syllabus/pair-programming/` and submit the `PARTNER` file as required. Pair programming is encouraged in all assignments.

Every file you submit should have as its first line your name and username. Its second line should be an RCS `$Id$` string in a comment. ***CAUTION:*** Before submitting anything, carefully read the section in the syllabus that covers cheating. For instructions to the grader, see the `.score` subdirectory. ***Verify your submit.***