

CS-256—Advanced Programming

Assignment 0, Due 17/Jan.

Jim Rogers

`jrogers@cs.earlham.edu`

Spring 2010

Objectives

This initial assignment is meant to be a warm-up. It includes a sequence of steps, each successively a little more complicated. You're not expected to be able to do all of them; that's OK. All of you are likely to be able to do at least the first one or two. The purpose is not so much to find out what you can do as it is to encourage you to recall what you can do. The last bit is open ended. If you get that you are entitled to be impressed with yourself.

There are quite a few steps, all told. If you find the early parts easy, it shouldn't be too hard to complete them all by Monday as long as you don't wait until Sunday to start. But you should not panic and spend all of the next few days working on this. Your grade is not going to depend on how many steps you get done, but rather on whether you get any steps done. On the other hand, you should not wait until Sunday night and then do the first part and call it good enough. You should genuinely try to complete as many parts as you can.

This program is also going to be our working example of procedural decomposition during the first couple of days of class.

Due

Monday, 17/Jan., at the *beginning* of class. As I don't expect you all to complete all of this, the normal extension policy doesn't apply. Do as much as you can. Wherever you get stuck, write a note to me saying what you got stuck on.

Overall goal

The idea is to write a calendar program similar to the unix `cal` command. (Try it: type `cal 1752` at the command prompt.) The main difference is that, unless you do the very last part, you're not going to have to figure out what day of the week the year starts on and we're going to print the months in a single column.

1 Printing one week

1.1 Version 1

The first step is to write a very short program `weekV1`. (The source file should be named `weekV1.cpp`.)

This program reads two numbers from `cin`:

1. An initial day of the week, given as a number between 0 (Sunday) and 6 (Saturday).
2. An initial date, given as a number between 1 and 31.

It then writes, to `cout`, a single line of a calendar (one, possibly partial, week) which starts with enough spaces to line up with the initial day of the week and then prints the dates of the days between that day and the end of the week. Don't worry, at this point, if the week runs past the end of the month. (Assume, for now, that this is a non-Terran calendar which has months with more than 31 days.)

Finally, on the next line, it writes the next date (the date the day following the last day printed, which also doesn't have to be a valid date), leaving `cout` at the beginning of the next line.

Examples:

If the initial day is 0 and the initial date 17 you should get something like:

```
17 18 19 20 21 22 23
Date: 24
```

If the initial day is 2 and the initial date is 28 you should get something like:

```
28 29 30 31 32
Date: 33
```

Note that this prints Tuesday through Saturday (the hypothetical 32nd day of the month).

Lining the numbers up in columns:

Each date is going to be either one or two digits (zero if you count the spaces before the first day of the week), so each column is going to be three characters wide. Also, we want the one digit numbers to align with the righthand side of the column. The simplest way to do this in C++ is to use *I/O manipulators*. To print the next number in two columns (if possible) you push `setw(2)` into `cout`. To make it align on the right you push `right` into `cout`. For example:

```
cout << setw(2) << right << date;
```

will print the value of `date` in two columns, aligned with the right side of the column.

Note that the format is reset each time you push an actual value into `cout`, so you have to repeat the two I/O manipulators for each number. (This will not be much of a annoyance for you if you have a good structure for your program.)

In order to use I/O manipulators, you have to include the `iomanip` header file. So the top of your program should include the two lines:

```
#include<iostream>
#include<iomanip>
```

(Also, don't forget `using namespace std;`)

1.2 Version 2

The next step is to bring the calendar down to earth.

The program `weekV2` reads *three* numbers from `cin`:

1. An initial day of the week, given as a number between 0 (Sunday) and 6 (Saturday).
2. An initial date, given as a number between 1 and 31.
3. The number of days in current month, given as a number between whatever initial date was given and 31.

It then writes, to `cout`, a single line of a calendar (one, possibly partial, week) which starts with enough spaces to line up with the initial day of the week and then prints the dates of the days between that day and either the end of the week or the last day in the month.

Finally, it writes the next date (the date the day following the last day printed, which doesn't have to be a valid date if the last day printed was the last day of the month) and the next day of the week (the day following the last day printed, which *must* always be between 0 and 6).

Examples:

If the initial day is 0, the initial date 17 and the number of days in the month is 31 you should get something like:

```
17 18 19 20 21 22 23
Date: 24, DoW: 0
```

Note: If the week ends on Saturday the next day of the week is 0 (Sunday), not 7 (the day after Saturday in the new metric calendar).

If the initial day is 0, the initial date 27 and the number of days in the month is 31 you should get something like:

```
27 28 29 30 31
Date: 32, DoW: 5
```

Note that this prints Sunday through Thursday and prints nothing (or spaces if you like) for Friday and Saturday.

If the initial day is 2, the initial date is 28 and the number of days in the month is 31 you should get something like:

```
28 29 30 31
Date: 32, DoW: 6
```

Note that this prints Tuesday through Friday, leaves spaces for Sunday and Monday and prints nothing (or spaces if you like) for Saturday.

1.2.1 Version 3

Finally, spiff this up with some column headings.

The program `weekV3` works exactly the same as `weekV2` except that it writes the line:

```
S M T W T F S
```

to cout before the week, so that the columns are headed by the day of the first letter of the day of the week.

Examples:

If the initial day is 2, the initial date is 28 and the number of days in the month is 31 you should get something like:

```
S M T W T F S
    28 29 30 31
```

Date: 32, DoW: 6

Using a char array:

It is, of course, possible to print the column heading as a single string. But I would like you declare an array of `char`, with one character for each day of the week. (So position 0 of the array is 'S' as is position 6.) Recall that you can declare and initialize a constant array of characters named "names" with:

```
const names char[] = {'S', 'M', 'T', 'W', 'T', 'F', 'S'};
```

It is not necessary to give the size of the array if you are initializing it. The compiler can figure it out.

A string is just an array of `char`, so you can also declare `names` as:

```
const names char[] = "SMTWTFS";
```

This will actually give you an array with eight slots instead of seven. There is a non-printing character marking the end of the string. But this is not going to matter here.

2 Printing one month

The idea now is to abstract your `weekV2` program (not `weekV3`, that's for later) as a function and then call it repeatedly to print each successive week of a month.

2.1 Version 1

`monthV1` reads two numbers from `cin`:

1. the initial day of the week (between 0 and 6)
2. and the number of days in the month.

and prints the entire month, followed by a blank line. It should do this by calling a function `week` once for each week until the entire month has been printed. Finally, it writes the next

date to `cout`, leaving `cout` at the beginning of the line.

Turning a program into a function: The program `weekV2` read three numbers: a day of the week, a date and the number of days in the month, from `cin`, printed one week of the calendar and then printed two numbers: the next date and the next day of the week.

We can pass the day of the week, the date and the number of days in the month into a function as arguments. Functions, properly only return a single result. Initially, we'll just return the next date—we won't need the next day of the week yet.

So you can declare `week` as follows:

```
int week( int dow, int date, int days )
{
    ...
    return(date);
}
```

Example:

If the initial day is 2, and the number of days in the month is 31 you should get something like:

```

      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
```

Date: 32

2.2 Version 2

When we print the whole year, we are going to know the day of the week the next month starts on, not the date (which really ought to be 1). But the only place we know the next day of the week is in `week`. So we are going to have to return *both* the next date *and* the next day of the week—two values, not one. (For the mathematicians among you, this makes for a function that doesn't look much like a function.)

The standard way to do this is to pass one of the values back via a *reference parameter*, i.e., an argument that is passed by reference, not by value. Then the argument is updated in the function, the update will be made for the actual parameter in the program that called it.

To do this, all you need to do is to change the signature of `week` to be

```
int week( int& dow, int date, int days )
```

Now `dow` will be both an input and an output parameter. When we update `dow` in the function, we will be updating the actual variable in the calling program that we passed in as the first argument of `week`.

The program `monthV2` should use this to output the next day of the week instead of the next date after printing the month.

Example:

If the initial day is 2, and the number of days in the month is 31 you should get something like:

```

        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
```

DoW: 5

2.3 Version 3

Finally, `monthV3` should print the column header as well as the month.

Example:

If the initial day is 2, and the number of days in the month is 31 you should get something like:

```

    S  M  T  W  T  F  S
        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
```

DoW: 5

3 Printing the entire year

OK, now we're ready to print an entire year.

3.1 Version 1

Program `yearV1` should read a single number from `cin` the day of the week for January 1st. It then prints each month for the entire year.

You will need to abstract your `monthV3` (might as well keep the header) program into a function named “`month`”. Do this in the same way that you abstracted `weekV2` as `week`. (What value does `month` need to return?)

The main question for `yearV1` is getting the right number of days in each month. You should do this with an array of `int`, one for each month of the year.

3.2 Version 2

Next, we'd like to print a month name header at the top of each month. You might do this initially with an array of `char`, printing just the first letter of each month, just like you did for the days of the week. But once you have that working, I'd like you to do this with an array of strings.

Declaring an array of strings: Once we look at how arrays are actually implemented in C++, later in the semester, we'll come back to this example and look at how this works. But for now, we're going to use a bit of unexplained boiler-plate. To declare an array of three strings and initialize them to "One", "Two", "Thr":

```
char* names[] = {"One", "Two", "Thr"};
```

If you then push `names[2]` into `cout`, "Thr" will be printed.

Modify `yearV1` so it prints the name of the month at the top of each month.

3.3 Version 3

OK, if you've made it this far, here's where it gets challenging. In order to print the calendar for arbitrary years, we need to know two things:

- The day of the week of January 1st
- Whether February has 28 days or 29.

You might start with modifying `yearV2` to print 29 days in February in leap years. The program `yearV3` will read a single number from `cin`, the year, and will print a calendar with the right number of days for each month, but with January 1st always being a Sunday (or a Wednesday, I suppose, if you wish). This part is not too hard. You just have to find out how to tell if the year is a leap year and then figure out how to use that information to get the right number of days in February.

The next part is a lot harder. `yearV4` should read a year from `cin`, figure out which day of the week it starts on and then print the correct calendar for that year. Do I know how to do this? No. But I do know how I would find out. Hint, I wouldn't try to do it from first principles. Somebody has figured it out, I know, because `cal` seems to work.

Assignment

Please submit the following things:

1. Your source code for each program you complete.
2. An example of the output of the most capable program you have completed. This should demonstrate that the program works properly, so it may need to show the result of running the program several times. (If you don't know how to do this, ask and we'll go over it in class.)
3. A brief statement of what you got stuck on, not which program you got stuck on (I can figure that out) but what the program was supposed to do that you couldn't figure out how to do. You can just write this on the same page as your output.

Make sure that you submit each of the things listed above and that each page is clearly marked with your name.