You are viewing this thread in readonly mode.

Project1, due Mon Feb 22 at 11:59pm Chicago time #295

A Adam Shaw STAFF
a year ago in General

2,721
VIEWS

This year's project will involve creating a calendar with event-planning features. The project will unfold over a series of homework-like assignments (currently planned to be three). The first project is directed at drawing calendars and integrating them into an interactive universe. Please read this whole document before you begin to write code.

This is not a team project and you must work independently on it, consulting neither current students nor former students, as students have done some similar, though not identical, work in past quarters. This year's calendar will be more advanced than anything attempted in CS151 prior to this year.

For the first phase of this project, you must be able to render a month calendar according to a detailed format, and move through months interactively in a universe program. The basic data structures involved in this work are the `CalFormat` and the `CalWorld`:

```
(define-struct CalFormat
  ([cell-size : Integer]
   [title-bar-bg : Image-Color]
   [title-bar-font : Image-Color]
   [title-bar-height : Integer]
   [day-label-bg : Image-Color]
   [day-label-font : Image-Color]
   [day-label-height : Integer]
   [cell-bg : Image-Color]
   [cell-font : Image-Color]))

(define-struct CalWorld
  ([format : CalFormat]
   [current-month : Integer]
   [current-year : Integer]))
```

A `CalFormat` struct contains colors and sizes in pixels determining the appearance of a calendar. The individual items are described in more detail at the bottom of this document. You need to write a function as follows:

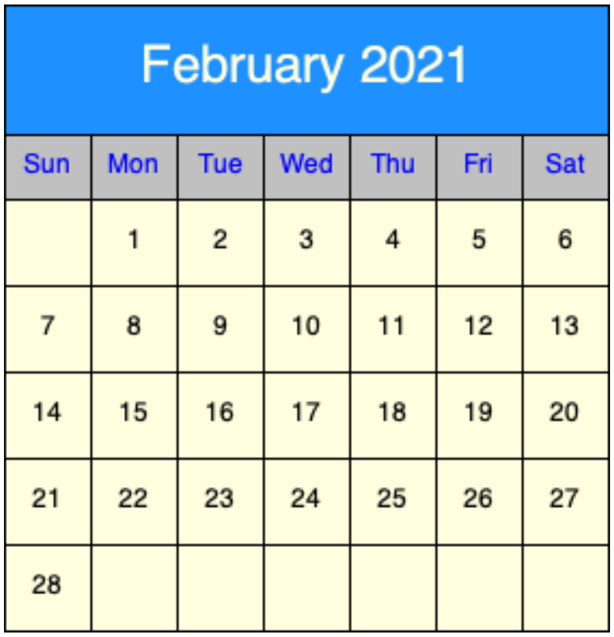
```
(: draw-month : CalFormat Integer Integer -> Image)
```

where the two integers are month and year, respectively. For the month integer, 1 corresponds to January, 2 to February, and so on up to 12 for December.

Here is an example format:

```
(: fmt0 CalFormat)
(define fmt0
  (CalFormat 40
             'dodgerblue 'lightyellow 60
             'silver 'blue 30
             'lightyellow 'black))
```

and here is February 2021 rendered with that format by calling `(draw-month fmt0 2 2021)`:



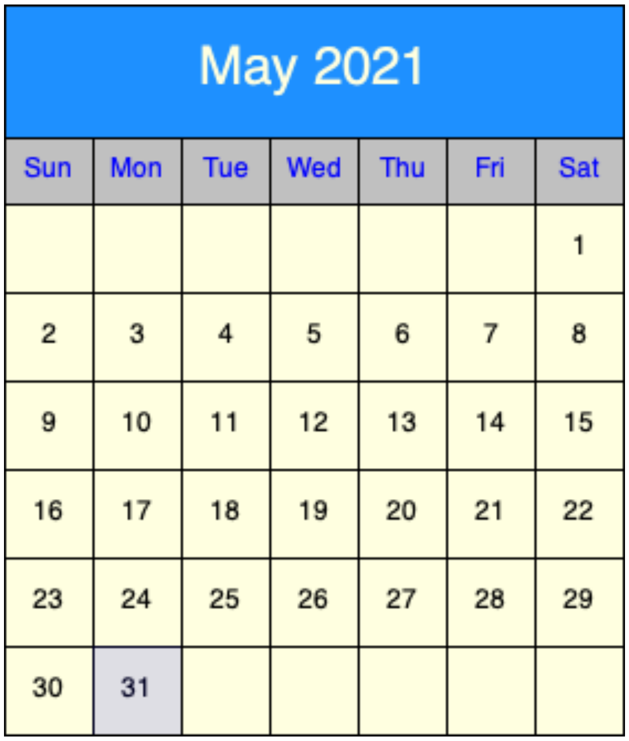
February 2021

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	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						

The calendar must render correctly given any format (within reason); that is, it must be flexible with respect to the sizes that are determined by the given `CalFormat`, and rely on no hard-coded lengths in its rendering. We say "within reason" because if the program is given a format with cells that are 1x1 pixel and with black-on-black text, for example, and the calendar looks awful, that definitely falls under GIGO, and your code doesn't have to be written to protect against it.

Note that the calendar starts with Sunday at the left edge, and both the first and last rows of days are filled out with empty cells as needed. Your calendar drawings must meet these constraints too.

There is a slight twist to the calendar rendering not yet mentioned, which is as follows. Here is May 2021:



May 2021

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						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					

(Technical note: Ed seems to be changing the sizes of both these images very slightly in the presentation, so please don't take them literally pixel for pixel.)

You will notice that May 31 appears differently than the other days. That is because the last Monday in May is Memorial Day in the United States. You must provide a visual indication of Memorial Day when you render May. It can be a letter M, it can be a difference in coloring, as shown here, or any other visual feature you choose, but there must be some visual distinction applied to the last Monday in May to indicate Memorial Day. Similarly, you must provide visual indications of Labor Day (the first Monday in September) and Thanksgiving (the fourth Thursday in November). In all three cases, we are not dictating to you what the form of the visual distinction is, just that there is one.

Drawing a month correctly depends on calculating days of the week, which you did back in lab2; you are explicitly permitted to include as much of your lab2 code in project1 as you see fit (and yes, we're aware you collaborated on lab2 with one or two partners). Just be clear about where the code came from if you include it. Borrowing from lab2 may also entail including various date-related data structures, which you may also do. In general, you may include whatever additional data structures you like in this work, as long as the `CalFormat` and `CalWorld` structs defined above are present and intact as they appear above and unaltered.

When your `draw-month` function is up and running, you can consider moving ahead to the universe part of the project. Write a keyboard-handling function that reacts to the arrow keys as follows:

- the right arrow key (which appears as `"right"` to universe) moves to the next month
- the left arrow key (which appears as `"left"` to universe) moves to the previous month
- the up arrow key (`"up"`) moves forward one year
- the down arrow key (`"down"`) moves back one year

There will be a demonstration of a working application in the Panopto video stream for this course, which will allow you to observe how this is supposed to work.

Please note when the world is drawn, it must be drawn large enough to hold a maximum-size month. You will notice May 2021 (see above) spans six weeks. This is the most a month can ever span. Make sure that when you draw the world, there is always enough space to present a six-week-spanning month; otherwise, part of the calendar might be cut off when you advance to such a month.

Finally, you need not worry about the fact that certain months will present incorrectly, falsely suggesting, for example, that there was a Labor Day in September of 1654 (not to mention the very calendar itself has undergone various critical revisions throughout history). We will worry about sensible dating constraints later in the project series, but not yet.

More than in any other project, this project provides the opportunity to think about designing useful functions on your own, without so much guidance from us. You ought to invest a lot of effort in thinking about which functions to write before writing them. The only functions we are specifying for this project are

```
(: draw-month : CalFormat Integer Integer -> Image)
```

and

```
(: run : CalFormat Integer Integer -> CalWorld)
```

both of which take a format, a month, and a year, in that order. Everything else is up to you. As a rough benchmark, my own (Shaw's) prototype implementation of this project contains around 30 functions. You might have more or fewer functions than that in your implementation, but my number gives you an idea. Most of the functions I wrote are short and directed at one particular task, which is the way I prefer to organize my work (so that it remains tractable and manageable). Well-chosen function names and well-written purposes matter a lot. You need not write tests for functions that produce complex images or run worlds, but for the rest, you do, as usual.

Finally, there is one last thing that's important to say. You might notice that there are only finitely many calendar arrangements that are possible. That is, there are seven possible start days to a month, and four possible month lengths (28, 29, 30, or 31 days). Therefore, there are, in the realm of possibility, only 28 different calendar arrangements you might have to draw, ever. It is critical that your code not simply enumerate all 28 possibilities, because that is a) supremely tedious and b) sidesteps important critical thought on the programmer's part. Be sure your code uses logic to determine the calendar format, and does not consist of an exhaustive enumeration of all the combinatorial possibilities of calendar drawing. Be forewarned that exhaustive enumeration code will not earn any credit, because we consider it so important to dissuade you from writing code like that.

So, in conclusion, you must build a universe program that draws months according to a specified format, allows navigation through months with the arrow keys as specified above, and visually distinguishes Memorial Day, Labor Day, and Thanksgiving. When you've done that, you've done project 1. Please submit your work in `project1/project1.rkt` in your git repository.

As promised above, here is the `CalFormat` struct in more detail.

```
(define-struct CalFormat
  ([cell-size : Integer]
   [title-bar-bg : Image-Color]
   [title-bar-font : Image-Color]
   [title-bar-height : Integer]
   [day-label-bg : Image-Color]
   [day-label-font : Image-Color]
   [day-label-height : Integer]
   [cell-bg : Image-Color]
   [cell-font : Image-Color]))
```

- `cell-size` is the size of each square day cell in pixels
- `title-bar-bg` is the background color of the title bar (the area that says "February 2021")
- `title-bar-font` is the font color of the title bar
- `title-bar-height` is the height in pixels of the title bar; the width is seven times the cell size
- `day-label-bg` is the background color of the day label row (where "Sun", "Mon", etc.,

appear)

- `day-label-font` is the font color of the day label row
- `day-label-height` is the height in pixels of the day label row
- `cell-bg` is the background color of the day cells
- `cell-font` is the font color of the day cells