Project 3, due Tuesday, March 16 at 11:59pm #474

A Adam Shaw STAFF
a year ago in General

Greetings, everyone! It's time to get going on the final piece of CS151 coursework.

Project3 is due Tuesday, March 16 at 11:59pm. You should save it in `project3/project3.rkt` in your repository and we'll collect it from there as usual.

You should start getting ready for project3 by watching the video in the Panopto stream; it's called "lecture25.1 03-09 Project3 preview". The video makes clear enough what the result of the project ought to be. We suggest you refrain from writing any code until having viewed the video and having read through the whole description that follows.

Project3 is built on the following data structures:

```
;; ===== data definitions

(define-type (Optional A)
  (U 'None (Some A)))

(define-struct (Some A)
  ([value : A]))

(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 Time
  ([hour : Integer] ;; from 0 to 23
   [minute : Integer]
   [second : Integer]))
```

```
(define-struct Date
  ([month : Integer] ;; 1 for January, ..., 12 for December
   [day : Integer]
   [year : Integer]))

(define-struct Span
  ([start : Time]
   [end : Time]))

(define-struct Event
  ([date : Date]
   [time : (U 'all-day Time Span)]
   [description : String]))

(define-type EventTree
  (U 'Empty EventNode))

(define-struct EventNode
  ([date : Date]
   [events : (Listof Event)] ;; maintain this list in ascending order
   [lsub : EventTree]
   [rsub : EventTree]))

(define-struct CalWorld3
  ([mode : (U 'calendar 'help 'entry)]
   [entry-mode : (U 'start 'end 'description)]
   [format : CalFormat]
   [calendar-current-date : Date]
   [now-date : Date]
   [now-date-string : String]
   [now-time : Time]
   [notepad : String]
   [opt-start : (Optional Time)]
   [opt-end : (Optional Time)]
   [events : EventTree]))
```

Note there are three different kinds of events, reflected in the way their time is stored: these are the all-day event, the momentary event, which happens at a particular time but has no duration, and the span event, which has a start time and an end time. These three kinds of events are represented by the union type attached to the event's time field. Events are not allowed to cross from one day to the next, but your code need not enforce this property.

Events are ordered as follows. (***Clarification****: the following discussion assumes the two events' dates are the same. An earlier date* ***does*** *mean an earlier event.*) All-day events are ordered

before timed events. All-day events among one another are ordered by comparing their descriptions with `string<?`. Timed events -- momentary events and span events -- are ordered by start time. Momentary events are ordered before span events when the momentary time is the same as the span event's start time. A span event is earlier than another span event according to the start times, and the end times are compared to break ties. If any two momentary events or span events are identical with respect to times, ties are broken by the string order (with `string<?`) of the descriptions.

The `CalWorld3` struct has the following fields. The mode field has a new mode called 'entry, and there is an entry sub-mode for 'start, 'end, and 'description. These modes correspond to entering the start time, end time, and description of an event, respectively. When the mode in the world is not 'entry, it doesn't matter what the value of entry-mode is.

The format, calendar-current-date, now-date, now-date-string, and now-time are all as before in project2.

The world's notepad field is a place to store the text that is currently being typed in entry mode. If the user is in the process of entering, for example, "9:00am", the notepad will contain, after each keystroke, "9", "9:", "9:0", "9:00", "9:00a", and finally "9:00am". Only when the notepad reads "9:00am" is it ready to be given to the string->time function (see below). The notepad is there to contain the string at every intermediate stage.

The fields opt-start and opt-end store the start time and end time the user has entered for a given event, and they are 'None when they contain no current information.

Finally, the events tree stores the accumulation of all events that have been entered in a given run of the program.

You have freedom to write the project3 code as you see fit, for the most part, but the following operations are mandatory and we will look for them:

```
: string->time : String -> (Optional Time))
;; convert a string like "11:25am" to a Time struct, if possible
;; ex: (string->time "10:00am") -> (Some (Time 10 0 0))
;; ex: (string->time "4:30pm")  -> (Some (Time 16 30 0))
;; ex: (string->time "abcde")   -> 'None


(: event<? : Event Event -> Boolean)
;; determine if the first event is "less than" the second according to event order


(: insert-event-tree : Event EventTree -> EventTree)
;; insert an event into an event tree, creating the node if needed
```

```
;; note: no duplicate check is necessary; insert the event no matter what
;; note: maintain the events list in ascending event order

(: insert-event-world : Event CalWorld3 -> CalWorld3)
;; insert an event into the event tree in a cal world

(: retrieve-events : Date EventTree -> (Listof Event))
;; fetch the list of events for the given date
;; return empty list if that date is not present in the tree

(: run : CalFormat Integer Integer -> CalWorld3)
;; run the application, given format, month, and year
```

Note: there will be video content coming soon where we will talk about `string->time` in some detail. We have not yet worked with individual characters, which are needed in `string->time`, and the video will address that.

(***Clarification****: the only formats that must be supported by `string->time` are H:MMam, HH:MMam, H:MMpm, and HH:MMpm. You may support other formats if you like.*)

In calendar mode, the application must display the events alongside the calendar, roughly (but not necessarily exactly) as shown in the video demonstration. If you have a different idea about how the enumeration of events should look, by all means, go ahead and try it; just make sure that for a given event all of its information is clearly visible. Events must appear in event order (per `event<?`) top to bottom. You do not need to account for very long lists of events that might extend beyond the lower boundary of the world window.

The help mode can be updated to say something about event entry if you choose, but we will not evaluate your work on help mode for project3. It still has to work (i.e., not crash the application) with the same keystrokes, but that's it.

Event mode must be triggered by striking the return key, which can be matched with the string `"\r"` in your key-handling code. The event the user is entering occurs on calendar-current-date. Once in event mode, the user should have a way to enter the start time for the event. Please note that DrRacket's universe has no notion of a "text box" for data entry, and what you see in the video is just ordinary rectangles and text being drawn on the screen; all the data entry is achieved with an ordinary react-to-key function and corresponding manipulation of the notepad field in the world structure. Use the notepad field of the CalWorld3 structure to build a string as the user enters one character at a time, until they strike return (`"\r"`), at which point you should apply string->time to the contents of the notepad and store the resulting Time in the opt-start field of the world. Once the user has entered a start time, the

application can clear out the notepad and proceed to the 'end sub-mode, and the user should be able to enter the end time similarly. After the end time is entered, proceed to description mode, build up the description in the notepad, and, finally, when the user strikes return one last time, build the event and store it in the tree in the world structure.

Consider these corner cases as well. If the user enters no start time at all, you should advance directly to the description mode (skipping over end mode) in order to create an all-day event with whatever description the user types. If the user enters a start time but no end time, create a momentary event (with a `Time` instead of a `Span`).

If the user strikes the escape key while in entry mode, go back to calendar mode and discard any partially-complete event information immediately. If the user strikes the backquote key while in entry mode, clear out the notepad field and otherwise continue operation.

Finally, in order to make sure DrRacket/universe doesn't add strings like `"shift"` to the notepad when the user strikes the shift key, test the user's keystroke in some simple way (e.g., `(= (string-length key) 1)`) before appending them to the notepad. If you wanted to be very paternalistic about things, you could allow only legitimate keystrokes (digits, colon, a, m, p) during time entry, but we won't insist on that.

It is difficult to put into words what this code ought to do (having just attempted to do so), but I will assert that the data structures themselves give you substantial guidance when you set out to write the code. The BST routines can be written entirely independently of any other code, as can other parts, so I suggest you start on this exercise by approaching one part at a time, and not necessarily worrying about other independent parts. Do start early and give yourself plenty of time to reflect and step away from the screen in the process of building this application.

For those of you wondering, we will release a project2 reference implementation some time soon (not immediately). The special distinctions of Thanksgiving, *etc*. from project1 are also not part of project3.

Thanks for you efforts, and perhaps we'll see you in office hours in our remaining week or so.