CS 1112 Spring 2020 Project 5 due Thursday, April 30, at 11:00 PM EDT

You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, "you" below refers to "your group." You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student's code and it is certainly not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

Objectives

Completing this project will help you learn about cell array, strings, objects, and working with data files.

Ground Rule

As usual, use only the functions and constructs learned so far in the course.

1 Interactive exercise log

Do you keep an exercise log? If you like to run (and aren't currently sheltering in place), do you know exactly how long your route is? How many miles per month do you run and what is your typical pace (minutes/mile)? These days apps like Strava can track this information, but in this project you will create a MATLAB application to keep a simple running log. You'll be able to enter your run data—at least the estimated distance and time and optionally the path information—and review the statistics. Two sample graphics are shown below.



The user will interact with a text-based menu in the Command Window to enter text or numeric data (date, time, etc.) and can choose to use the graphical interface (just a figure window) to "click in" a route in order to determine the distance. Download the file p5files.zip which contains function skeletons as well as data files from the course website. Run the program to see the menu:

RunLog('RunData1.txt', 'IthacaNETrails.jpg')

The code for the menu system has been completed for you so you can enter your choice (enter 1, 2, 3, or 4), but the only choice that actually works right now is 4: Exit. You will complete the functions that will make the whole thing work. Read the given code in RunLog.m now to make sure that you understand how the menu system works. Note the use of the built-in function isempty(). isempty(x) returns true (1) if the variable x is an empty array of any type; otherwise it returns false (0). Later you will implement the last subfunction in the file, but don't worry about it just yet.

1.1 Modeling runs as Events

If we think about the kinds of objects we'd interact with in an exercise log, a log entry (corresponding to a particular run) is a natural candidate. We'll call a single entry an "Event", and we've provided an Event class specifying the data and behavior of these objects. An Event has seven properties, as described by the comments in the classdef. The date, distance, and duration of the run must be supplied when an Event is constructed. Additionally, map data and route coordinates can be added to an existing Event object by invoking its methods.

One of the methods, pace(), has not yet been implemented. Instead, the body of the function contains:

These lines represent a "dummy statement" known as a "stub" and do not accomplish the function's specification. Instead, they invoke the return keyword, which causes the function to end (return) immediately at that point in the code. All of the skeleton functions in the given code are stubbed like this in order to allow the main function RunLog() to execute without a runtime error (as you saw earlier), even though the required functions have not been implemented as specified yet. Stubbed functions allow us to build a big software project one function (functionality) at a time given the overall design specified by the collection of function headers. Planing out your development with stubs is a great way to facilitate top-down design. Be sure to remove the stub when you work on the function.

To calculate the pace (minutes/mile), you need to first convert the time property (a string, e.g., '31:30') to a numeric value (e.g. 31.5, in minutes), then divide by the distance property. The function str2cells() is given to help you with this and other upcoming tasks. Read function str2cells() and make sure that you understand it. Try calling it on some example strings in the Command Window; for instance:

```
c= str2cells(' CS 1112 spring14', ' ')
d= str2cells('hey there! what?!', '!')
```

Complete the pace() method as specified, replacing the stub with your own code.

1.2 Log a run with no route

Function newRun() allows a user to enter the following data: date, distance, and time. The code for prompting the user to enter values is given in the skeleton file already. Notice the use of a second argument in the input function:

```
theDate= input('What is the date? ', 's');
```

The second argument, 's', says that the expected input is a string, so the user does not need to type the quotation marks around the string when entering the string data during program execution, i.e., type 3/3 instead of '3/3', for example.

Complete newRun() as specified (see the comments in the file). You'll need to call Event's constructor with the appropriate arguments. Test newRun() before moving on! For example, call it in the Command Window like this:

myEvent= newRun();

Example Command Window interaction and output is shown below:

```
What is the date? 3/20
What distance (in miles) did you run? 3
How long did it take? Enter as <minutes>:<seconds> 29:10
   Recorded data and statistics
   -----
Distance: 3.0 miles
   Time: 29:10
   Pace: 9.7 minutes/mile
   Date: 3/20
```

Now see what is in variable myEvent by typing disp(myEvent). You should see this output in the Command Window:

1.3 Log a run with a route

Function newRunMap() allows the user to click on a map to indicate the route:

```
function EV = newRunMap(mapName)
% Return an Event object (EV) based on user-provided data input
% interactively on a map (image file named by mapName) and from the Command
% window.
%
% 1. Prompt user for date and time (strings).
% 2. Display the map and calculate the scale.
% 3. Prompt user to click in the running route, displaying the path and
% the distance as the clicks are made.
% 4. (optional) "Highlight" the path by "graying out" the map outside of a
% bounding box for the path.
% 5. Create the Event object.
% 6. Display in the figure title the statistics, including the pace.
```

The function code is partially given. Read the given code and comments carefully when adding/modifying code. Here are some additional notes:

- 1. Map scale. The code for this calculation is given. On the given map image the magenta 2-pointed arrow near the top covers one mile.
- 2. Route. As the user clicks on the map, both the route (lines on top of the map) and the distance (in the title) should be updated.

3. (optional) Highlight the route region. The given image is in color, so the image array (variable map in the code) is 3-d. We "highlight" the route region by "graying out" the rest of the map. Define a bounding rectangular box around the route. So for each pixel (i,j) outside of the bounding box, calculate the simple arithmetic average of the red, green, and blue values; this is the gray value of pixel (i,j) and should be assigned to map(i,j,1), map(i,j,2), and map(i,j,3).

Look closely at Figure 1 on page 1: the running route is red and outside of the bounding box of the route the image is grayscale. (We added a "buffer" 100 pixels wide to the bounding box for aesthetic reasons.)

Note in the given code that we use built-in function image (instead of imshow) for displaying the map image. This is because image accommodates multi-line titles and manual sizing of the figure window.

A hint on the order of doing things: after you change the RGB values in array map, use built-in function image (and the same axis formats) to display the revised map. Then plot the entire path on top.

Test newRunMap() before moving on! For example, call it in the Command Window like this:

```
myEvent= newRunMap('IthacaNETrails.jpg');
```

After you enter the date and time, click in the map scale, and click in the route, a figure similar to Figure 1 on page 1 should be the result. Then you can type disp(myEvent) to see what is stored in variable myEvent. Here is an example where the route is defined by five clicks:

Event with properties:

```
day: '3/31'
distance: 3.1808
   time: '43:58'
mapName: 'IthacaNETrails.jpg'
   scale: 0.0015
        x: [673.0239 502.4948 950.1335 1.2102e+003 696.4716]
        y: [319.2316 698.6587 832.9503 120.9916 293.6523]
```

1.4 Writing to and reading from the data file

- 1. Complete subfunction writeData2File() in RunLog.m to *append* data to the data file. Read the first few "comment lines" of the example data file RunData1.txt to learn about the format in which the data should be written. In our data file the % symbol denotes a comment (not data).
- 2. Complete function ReadData() to read event data from the plain text data file and store the data in a cell array where each cell contains an Event object handle. This function assumes that the data is in the format as specified in the comments of our example data file RunData1.txt. The given code in the file checks whether the data file exists and takes care of the case where the file does not exist. (Built-in function fopen() returns -1 instead of the file identifier if the file to be opened for reading does not exist.)

Note: (1) Read the 1-page document "File I/O example" posted on the Projects page; use only the builtin functions discussed in that document for reading our data file. (2) The given function str2cells() may be useful again here.

1.5 Showing statistics

Function showStats() displays the data on all the recorded runs (in the data file) graphically. See Figure 2 on page 1 for example. The code to produce the graphics is given; you only need to write the code that computes the vectors for plotting. Read the given code and comments carefully.

Read the given graphics code to learn some new graphics commands! These are useful for producing professional looking graphics in your future work! Here are some of the techniques we have used:

- 1. Position and size a figure window using the figure property Position. In addition to the given code in showStats(), see Appendix A.13 (page 410) in *Insight*.
- 2. Control where tick marks are placed and what labels to use at each tick mark using the axis properties xTick and xTickLabel. In addition to the given code in showStats(), see Appendix A.2 (page 392) in *Insight*.
- 3. Show grid lines by using the command grid on.
- 4. You have used the command legend before. The given code for subplot 1 shows how to put the legend in a specific position; see also Appendix A.9 (page 404) in *Insight*. (In subplot 2 we let MATLAB choose where to put the legend.)
- 5. We made subplot 1 simple by using just one y-axis. Is it possible to have two y-axes, one shown on the left and one shown on the right, for the distance and cumulative distance, respectively. If you are interested in learning about this, check out the command plotyy using the *Help* facility in MATLAB.

Test your program! Execute RunLog() and test each menu option. Look at the resulting graphics and data file to make sure that everything is correct. Then try it out with maps of where you live. Happy (socially distant) trails!

Submission

Submit your files RunLog.m, Event.m, newRun.m, newRunMap.m, ReadData.m, and showStats.m on CMS (after registering your group).