Editor

Learning Goals

- Be able to comprehend data definitions for compound data.
- Be able to design functions operating on compound data.
- Be able to design world programs on compound data.
- Be able to use the on-key option to big-bang.
- Be able to use a wish-list to keep track of work remaining to be done in a large program design.

Introduction

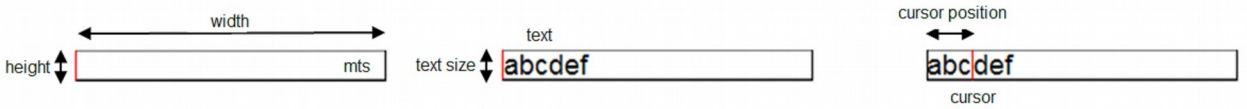
You will complete the design of a simple one line text editor similar to the one you see when you send a text message on your phone or type into the search bar on your browser.
We have started the design of this program, and you will complete it. You will see how the data definitions and wish-lists make it possible for you to finish a program someone else has started.
Once your editor is complete, starting it with (main (make-editor "abcdef" 0)) should display the following editor:

The red line is the cursor. In this text editor, you will be able to insert and delete text and move the cursor left and right. To help you with the project, we have made a demo video of what the program should do when it's done.

Main Project

One of the goals for this project is for you to get experience working with a larger program broken into multiple functions. At the very least, we would like you to have each case of handle-key call a different function to do that part of the work -- these are often called helper functions.

For example, in our solution, the first case of handle-key calls a helper function named move-cursor-left. So the answer for that cond clause is (move-cursor-left e). In our solution, those four helper functions also call two other shared helper functions. We want you to have at least four helper functions for handle-key. After that, the number of helper functions you use is up to you - but try to keep to a general guideline of "one task per function". Your experience with this will set us up for more detailed discussions on helper functions after the project.
Note that in the case of handle-key we are giving you part of the template, so you only need to complete the check-expects and code the function body.
Before you start programming, you should carefully review the domain analysis provided below.



Constant	Changing	big-bang options
width	text	on-key
height	cursor position	to-draw
text-size		
text-color		
cursor		
mts		

As you go through this domain analysis, be sure that you understand how we identified all of the constants, changing information and big-bang options. Then you should go through the starter file to ensure you understand the constants, data definition, and main function provided there. Use the data definitions and wish-list entries in the program to determine what you need to do to finish it.
If you're unsure about how to complete the functions, one suggestion is to go through the examples and draw images of the examples to help you understand the correspondence between Editor and what appears on the screen. In other words **follow the recipe!**
When you are done, the editor should implement at least the following functionality:

- The left arrow key should move the cursor left (unless already at left end of text).
- The right arrow key should move the cursor right (unless already at right end of text).
- The backspace key should delete the character before the cursor (if there is one).
- Key events of length 1 are normal characters, these should be inserted at the cursor position.

Note: You are not required to handle the DEL key on a PC. You only need to handle the "\b" key, which is labeled Delete on a Mac and Backspace on a PC. You only need to handle the Delete/Backspace key found on the main keyboard.
Don't worry about making a fully fledged editor! Don't worry about cases like the box getting too full. The goal is to have a simple design that works properly.

Downloads

1. starter file
2. demo video
3. subtitles

Image organizer/portfolio

Learning Goals

- Be able to represent arbitrary-sized domain information using a well-defined data definition.
- Be able to use design recipes for referential, self-referential, and mutually-referential data.
- Be able to design functions to render complex data.
- Be able to design functions to traverse and sort complex data.

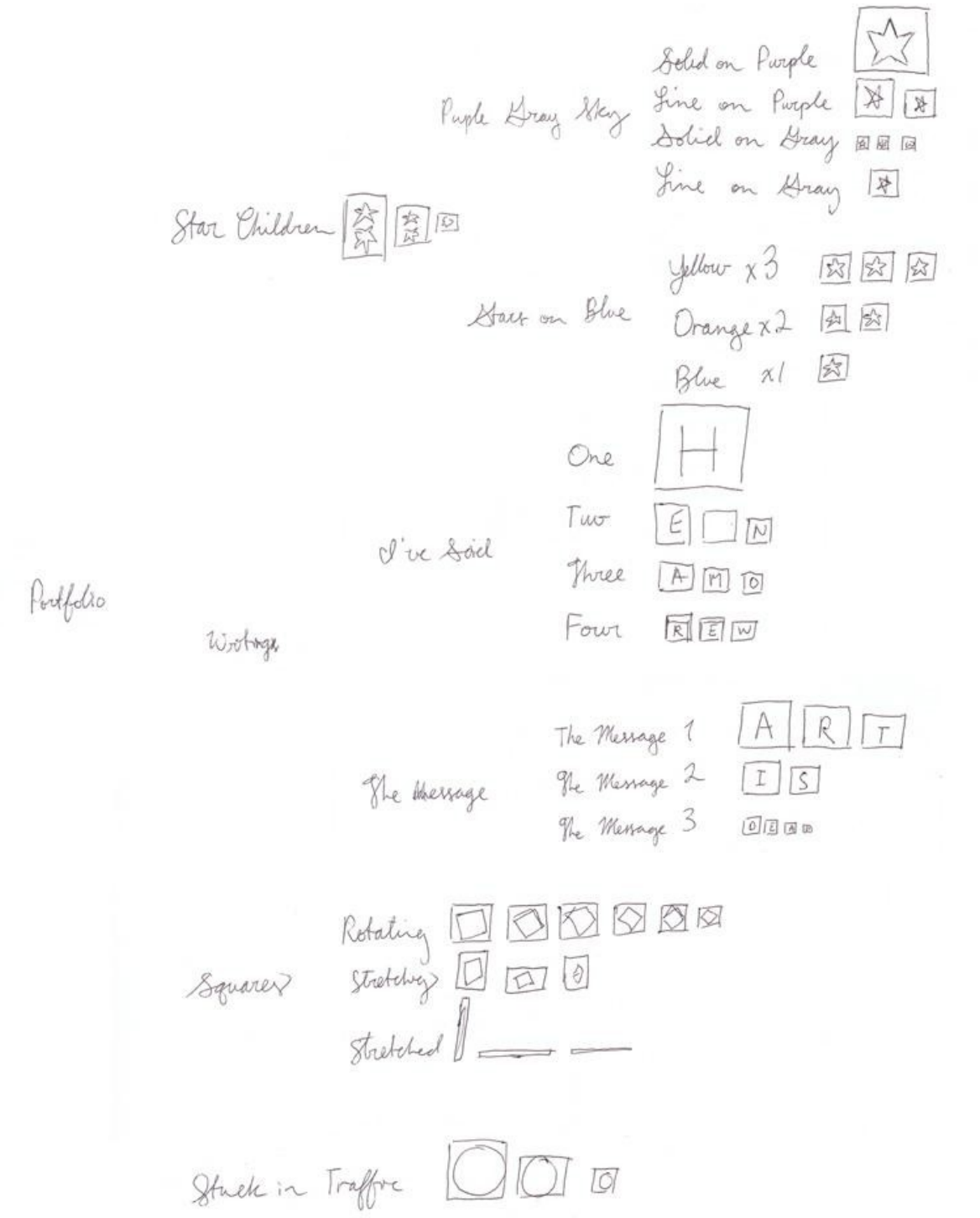
Introduction

You are an employee at the prestigious tech firm SPD industries, working on arbitrary-arity trees on a quiet Wednesday afternoon.

Your firm was recently commissioned to create a program for the renowned abstract artist Fan See, so that he could easily organize and store images.

In accordance with the abstract simplicity of his art, he wants a simple program which allows him to organize his images in named directories, which can contain an arbitrary amount of images and/or image sub-directories. The program should render the images in each directory in order from largest to smallest (by area), and render the directories in order from the one who has the largest sum of image areas to the one with the smallest sum of image areas. This will give him a quick idea of relative ink printing costs.

Your colleague Caroline has been working on the project. Unfortunately, Caroline got in what can only be described as a freak bike accident earlier today. Her backpack burst open and her laptop clattered to the ground and was completely destroyed by oncoming traffic. All the basic work on Fan See's portfolio structure has been lost, and the graphics department is anxiously waiting for it to make it look prettier for Fan See. Caroline's fine, of course, but under such shock she refuses to open a computer to recreate her program. You decided to ask her about it anyway. She sketches approximately how her program drew out Fan See's portfolio:



Since it's now your job to recreate the program, you get to work...

Your Task

Design an image organizer program.

It should include data definition(s) which accurately represent the information: named directories which can contain images and/or other directories. Each named directory can contain images, more directories, or both. Additionally, there should be no limit to how many images or sub-directories a directory can contain.

It should also include a function which draws the image organizer. This function should display the name of each directory, its images and its sub-directories, clearly indicating which sub-directories and images belong to that directory. The render function will also, before rendering, sort the images in each directory from largest to smallest, and sort the directories in descending order of total image area (of images contained in it and its sub-directories).

Finally, include a sufficiently complex data example so your grader can run your rendering function and see what it does. But remember, your function should be able to render any image directory corresponding to your data definition!

Important Note: *your data example does not have to replicate the sketch.*

Tips

- As this is a peer-graded project, value simplicity of prettiness in your render function. ***Adding embellishments*** such as lines indicating direction, braces, or boxes, ***IS STRONGLY DISCOURAGED***. While it may make the rendering a bit clearer, it will make your code ***substantially more complicated***.
- Our "prettiest" solution only includes spacing. You may include spacing as long as your code is clear; you might want to add a few annotations to it.
- Details such as the relative placement of images (beside directory name? below directory name?) and how exactly sub-directories are placed should not be important, but ***the structure of the information and sorting must be clear in the rendering.***
- If you want to get fancy, put a separate function design at the bottom of your file, that does the more fancy version. That way your grader can enjoy that version if they want to, without getting distracted by it when they grade the simple version.
-

Downloads

starter file

4 queens puzzle

This Final Peer Graded Problem will be your Final Exam for the course. We are using peer assessment for the final exam because we believe it is best able to assess your mastery of the course material. This project covers material from the whole course. You may use any material covered during the course, and you may use languages up to and including intermediate student. You must not have any require declarations in your file.

All the instructions for this project are here -- there is no starter file. So please read this text

several times carefully. The general rubric is below, with some grading precisions to be added later.

This problem involves the design of a program to solve the 4 queens puzzle. The key to solving this problem is to follow the recipes! It is a challenging problem, but if you understand how the recipes lead to the design of the Sudoku solver then you can follow the recipes to get to the design for this program. This problem is also easier than Sudoku. In particular, while you will end up needing to design a function like valid-board?, the version of that function needed for nqueens is much simpler.

The four queens problem consists of finding a way to place four chess queens on a 4 by 4 chess board while making sure that none of the queens attack each other. The four queens puzzle is one version of the more general n queens problem of placing n queens on an n by n board.

Here's what you need to know about the board and queens to solve the four queens problem:

- The BOARD consists of 16 individual SQUARES arranged in 4 rows of 4 columns. The colour of the squares does not matter. Each square can either be empty or can contain a queen.
- A POSITION on the board refers to a specific square.
- A queen ATTACKS every square in its row, its column, and both of its diagonals.
- A board is VALID if none of the queens placed on it attack each other.
- A valid board is SOLVED if it contains 4 queens.

There are many strategies for solving queens, but you should use a backtracking generative search that is trying to add 1 queen at a time to the board. So at each step you generate the next boards by finding all the empty squares, adding a queen to each empty square and discarding invalid boards. If the search reaches a valid board with 4 queens (a solved board) produce that result. So it's A LOT like the Sudoku solver. You may read online about other basic approaches, but again we would like you to use a backtracking generative search.

Some other notes:

- You should design a function that consumes a board - which will initially be empty - and tries to find a solution. Call your function queens. Your function should produce either a single solution if it can find one, or false if it cannot.
- This game has symmetry. We suggest you not try to take advantage of that fact, just design a function that produces one valid solution for a board (or false if it cannot find one).
- There are many clever data definitions one could use for the board, some take advantage of the symmetry of the solutions. Again, we recommend simplicity - just have your board represent every cell and whether it has a queen in it, or perhaps represent just the positions of the queens on the board. Don't get too fancy.
- You can tell whether two queens are on the same diagonal by comparing the slope of the line between them. If one queen is at row and column (r1, c1) and another queen is at row and column (r2, c2) then the slope of the line between them is: (/ (- r2 r1) (- c2

c1)). If that slope is 1 or -1 then the queens are on the same diagonal.

- We have described the problem as working for 4 queens. Feel free to make your design have a constant, SIZE, that controls how many queens it works for.

You may use the forums to clarify the problem and discuss the basic approach to solving the problem with backtracking search. You should not post anything on the forums that would actually appear in your solution, such as data definitions, function designs etc.

Finally, we want to note that we have used this problem before. So it would be fairly easy to track down a solution from the prior offering and submit that. That would of course improve your grade. But it would reduce your learning and leave you feeling badly about yourself. So don't do it. Solve the problem from scratch using the recipes. If you've taken the course before that's OK, just solve the problem from scratch using the recipes. If its easier for you this time that's fine -- presumably you took the course again in order to master the material better this time.

Have fun with this project! We feel this is a great project on which to end SPD Part 1. Its a perfect example of a project which is too hard for many introductory programming courses, but because of the design method you have learned is within your reach for this course. If you get confused then step back, slow down and follow the recipes.