Comp160
Final Design Worksheet

Spring 2018

Kinds of Data

If you take away one thing from this class it should be that programs designed so that the structure of the logic of their code mirrors that of the structure of their data are generally well designed programs. For that reason it is really, really important that you take the time to analyze the information for your program and determine an appropriate representation of that information as computational data. We encountered four main kinds of kinds of data this semester while making our foray into BSL Racket programming. Here is they are with some key sub-types of each kind where appropriate.

1. Primitive, Built-In Data Types
   (a) numbers
   (b) booleans
   (c) strings
   (d) images
2. Itemizations
   (a) Enumerations
   (b) Intervals
   (c) General Itemizations
3. Structures
4. Self-Referencing Data
   (a) List of ...
   (b) Natural Numbers

In this worksheet you'll begin a review of how the non-primitive kinds of data interact with key points in the 6 step design recipe. Towards this end, give some thought to the following situation:

> You're working on a media management program. Users of the program can track the audio and video media. You've determined that for audio you need to track the artist, title, length (in seconds) and genre. For video you need to track the title, length (in minutes) and director.

You'll be given data definitions and function templates for the data needed for this problem and asked to use those to write some functions.

1. Every data definition does two things: clearly states the relationship the data has to primitive built-in types or previously defined data types and a statement about how to interpret the data as real-world information. Circle or underline each part in each definition.

2. Now, let's look at this problem:

   We'd like to do some work with media lengths. Things like finding the longest or shortest file or adding up the total length of all the media in a list. To do this we need to compare audio to video. Towards this end we need to design a function named *TimeInMinutes* which takes as input a media data and returns the time of that media in minutes.

   (a) Provide the signature, statement of purpose and a stub for this function. Annotate what you wrote so that each of the three parts is clearly labeled.

   (b) You need at least two tests for this function. Why? Write those tests.

(c) Let's apply the media function template to this problem. We already have a function for getting the length of a video in minutes, the video length selector, fill it in for the implied helper. There does not exist a function for getting the length of audio data in minutes. Rather than explicitly computing that value let's create a wish list function called *TimeInMinutes/audio* that takes an audio structure and computes its length in minutes. Finish filling in your template with this function.

(d) (Optional) Complete the design of *TimeInMinutes/audio*. Begin with the signature, purpose, and stub. Then write tests. Then use the audio template to fill it out (we only need the one field selector though).

(e) Now let's do something more. Your user would like to filter their media collection by length. This means you need a function that takes a list of media and a length m (in minutes) and computes a list of media that contains on those media from the original list that are at least m minutes long. Follow the design process to develop this function. For step 4, the template state, be sure to begin from the list of media template and follow it along to other templates as needed just like we did in the previous problem.