CS61A Summer 2010

George Wang, Jonathan Kotker, Seshadri Mahalingam, Steven Tang, Eric Tzeng

Homework 3

Due: Monday, July 11 2010, at 7AM

Note: The number of stars (★) besides a question represents the estimated *relative* difficulty of the problem: the more the number of stars, the harder the question.

1 How to Start and Submit

First, download `http://inst.eecs.berkeley.edu/~cs61a/su10/hw/hw3.scm` and use it as a template while filling out your procedures. See `http://www-inst.eecs.berkeley.edu/~cs61a/su10/hw-faq.pdf` for submission instructions.

2 Data Directed Programming

1. The *University of California, Berkeley Library* is made up of a group of smaller libraries, each of which uses Scheme data structures to represent its collection. Unfortunately, each library has its own method of cataloguing books (using pairs, lists, nested association lists, etc.). Each library is required to store a book's title, author, and ISBN[1]. As the new head of IT for the centralized library system, you need to provide the librarians with a common method of accessing this data, and you decide to use Data-Directed Programming to accomplish this.

   You will be writing generic operators so that your program can work with any type of information. A generic operator is a procedure that can take in data of different types and perform its operation on any of these.

   For example, Moffitt Library implements its books as a list, like so:
   `((anna karenina) (leo tolstoy) 0143035002)`

   On the other hand, the Doe Library implements them as an association list. This means that they do them like this:
   `((title . 1984) (author . Orwell) (isbn . 0151010269))`

   We call each of these a Book Record. Note that we need different accessors to get to each element. For example, for Moffitt, they would make the following call:

   ```
   (define (author book)
     (cadr book))
   ```

   Doe, on the other hand, would make this call:

   ```
   (define (author book)
     (cdr (assoc author book)))
   ```

   (a) ★★ Implement a `get-book-record` procedure that, given a library's data file and book title, retrieves the corresponding book record. This procedure must work for any library's data files. Each Data File is a list of book records.

[1] `http://en.wikipedia.org/wiki/International_Standard_Book_Number`

Assume that each library has called `attach-tag` for each piece of data belonging to it. For example, Moffitt has called `(attach-tag 'Moffitt book-record)`. Furthermore, all the libraries have added all the information to a table. For example, Moffitt has called `(put 'Moffitt 'title caar)`.

(b) ★ Implement a `get-isbn-number` procedure that returns the ISBN from a given book record from any library's data file. How should the book record be set up for your procedure to work?

(c) ★★ Write a `find-book` procedure that, given a book title and a list of the book records of all of the libraries, retrieves the corresponding book record.

(d) ★ The Berkeley Public Library (on Shattuck Ave.) decides to join the UC Berkeley Library system. What do they need to do to put their database on the central system?

3 Message Passing

1. The procedures below implement a `dispatch on type` system for calculating the area and perimeter of shapes. This is also called Conventional Style. This is a reasonable solution for a system that is likely to encounter the addition of more methods/operators (eg. a `number-of-sides` procedure) than types (eg. triangles, rectangles).

```
(define pi 3.141592654)

(define (make-square side)
  (attach-tag 'square side))

(define (make-circle radius)
  (attach-tag 'circle radius))

(define (area shape)
  (cond ((equal? (type-tag shape) 'square)
         (* (contents shape) (contents shape)))
        ((equal? (type-tag shape) 'circle)
         (* pi (contents shape) (contents shape)))
        (else (error "Unknown shape -- AREA"))))

(define (perimeter shape)
  (cond ((equal? (type-tag shape) 'square)
         (* 4 (contents shape)))
        ((equal? (type-tag shape) 'circle)
         (* 2 pi (contents shape)))
        (else (error "Unknown shape -- PERIMETER"))))
```

But let's say that I want to add another type of shape, `triangle`. In order to implement this new shape, I would need to create a new constructor, `make-triangle`, and modify both the `area` and `perimeter` procedures. If we were to add more operations, this would quickly become a lot of work!

To solve this problem, let's try a different way of representing our data. Specifically, I want to define the area and perimeter procedures as follows:

```
(define (area shape)
  (shape 'area))

(define (perimeter shape)
  (shape 'perimeter))
```

Under the old Conventional Style, our shapes were represented as tagged data, and the tag told the `area` and `perimeter` procedures which formulas to use. However, under our new Message Passing style, our shapes now act as procedures that take a message (a word that is either `area` or `perimeter`) and return the appropriate value.

(a) ★ Re-implement `make-circle` and `make-square` in message passing style to work with the new `area` and `perimeter` procedures.

(b) ★ We want to make a new type, `e-triangle`, that represents an equilateral triangle. Add this new type to your message passing system. Your `e-triangle` should work with the given `area` and `perimeter` procedures, and you should NOT need to modify any of the code you've written for part 1a! *Note: The area of an equilateral triangle is given by* $\frac{s^2\sqrt{3}}{4}$.

2. In a two-dimensional rectangular Cartesian coordinate system, we can represent a point as a pair of x and y values.

(a) ★ Write a `make-rectangular-point` procedure that creates a `rectangular-point` in message passing style. A `rectangular-point` should accept the messages `'x` and `'y`, and return an error for any other message.

(b) ★ Create a new message, `distance`, that your `rectangular-point`s will understand. `(my-point 'distance)` should return a procedure that takes another `point` as an argument and returns the distance between the two `point`s.

(c) Points in a plane may also be represented in polar form, using a magnitude (r) and an angle (θ). Conversions from polar to rectangular coordinates use the equations:

$$x = r \cdot cos(\theta)$$
$$y = r \cdot sin(\theta)$$

★★ Write an analogous `make-polar-point` procedure that takes as input a magnitude and an angle. A `polar-point` should accept the `'x` and `'y` messages and return its corresponding rectangular coordinates. Any other messages should return an error.

(d) ★ Make a `rectangular-point` and a `polar-point`. Show how you would find the distance between the two `point`s, using your solutions to the previous parts of this question.

4 SICP Exercises

1. Number System Additions

(a) ★ 2.76

(b) ★★ 2.77

(c) ★ 2.79

(d) ★ 2.80

2. Coercion Exercises

The examples in the book use the `put-coercion` *and* `get-coercion` *procedures, which our version of UCB Scheme does not implement. You should be able to use the regular* `put` *and* `get` *procedures instead.*

(a) ★★ 2.81

(b) ★★ 2.83

5 Feedback

Now that you are done, please leave me some feedback at the following link regarding how the course is going. This is not worth points, but will give us valuable feedback that in turn improves your course experience. Thanks! `https://spreadsheets0.google.com/viewform?formkey=dDMtYndvOGFMbUVSc3dtcjlQUmFTVnc6MQ`