The Data Design Recipe

CS 5010 Program Design Paradigms
"Bootcamp"

Lesson 1.3

Learning Objectives for This Lesson

- By the time you finish this lesson, you should be able to:
  - list the steps in the data design recipe
  - list the pieces of a data definition
  - explain what define-struct does
  - write a constructor template and interpretation for simple data

The Data Design Recipe

1. What information needs to be represented in your program? What kind of information is each piece?

2. Structure Definitions

3. Constructor Template

4. Interpretation

5. Destructor Template

6. Examples

7. Review

DDR Step 1. What information needs to be represented?

- This is usually pretty clear; depends on the application.

- In general, information needs to be represented when it might take many values, and your program needs to know which one is right.

Example: representing a car

- How does one car differ from another?
- In a traffic simulation, I might only need to keep track of each car's position and velocity
- For TV coverage of an auto race, I might need to keep track of enough information to distinguish it from all the others in the race.
- For an auto dealer, I might need to keep track of enough information to distinguish this car from all the others in the world.

Example: representing a car

- Also need to keep track of other information that is needed by the application, of course.

DDR Step 2. Structure definitions

- In Racket, we represent compound data as a **`struct`**

- This is like a struct or record in other languages.

- For mixed data, we may need several structs
  - we'll see an example later

- In Racket, we define new kinds of structs or records with **`define-struct`**.

- We saw these in Lesson 0.4. Here's a review:

Example of a structure definition in Racket

`(define-struct book (author title on-hand price))`

Executing this **define-struct** defines the following functions:

make-book

> A constructor– given 4 arguments, returns a **book** with the given fields

book-author
book-title
book-on-hand
book-price

> Selectors: GIVEN: a **book**, RETURNS: the value of the indicated field.

book?

> A Predicate: GIVEN: any value, RETURNS: true iff it is a **book**

DDR Step 3. Constructor Template

- Tells how to construct a value of this type.
- We'll start with the most general case– mixed data– and then see how the others are special cases.

Remember our example of mixed data

In a wine bar, an order may be one of three things: a cup of coffee, a glass of wine, or a cup of tea.

- For the coffee, we need to specify the size (small, medium, or large) and type (this is a fancy bar, so it carries many types of coffee). Also whether or not it should be served with milk.

- For the wine, we need to specify which vineyard and which year.

- For tea, we need the size of the cup and the type of tea (this is a fancy bar, so it carries many types of tea).

Recipe for a constructor template

- Write down one **struct** definition for each alternative.

- Then write down the constructor template showing the type of each field.

- Example:

Struct definitions and constructor template for mixed data: example

```
(define-struct coffee (size type milk?))
(define-struct wine (vineyard year))
(define-struct tea (size type))
```
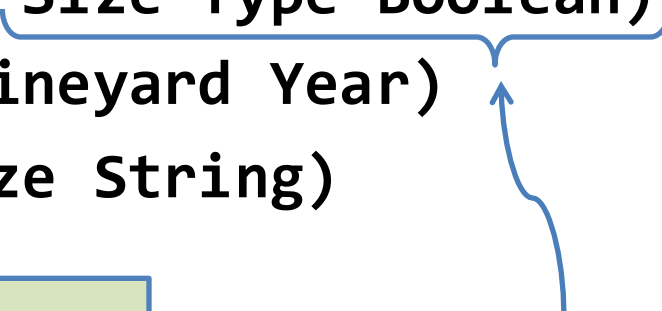
The structure definitions

```
;; A BarOrder is one of
;; -- (make-coffee Size Type Boolean)
;; -- (make-wine Vineyard Year)
;; -- (make-tea Size String)
```

The constructor template

The constructor template in more detail

```
;; A BarOrder is one of
;; -- (make-coffee Size Type Boolean)
;; -- (make-wine Vineyard Year)
;; -- (make-tea Size String)
```

Presumably **Size**, **Type**, **Vineyard**, etc. are data types defined elsewhere

The kind of data in each field

Example of a constructor template (compound data)

- For compound data, there's only one alternative, so you'd write one struct definition and have one line in the constructor template:

```
(define-struct book (author title on-hand price))

;; A Book is a
;;   (make-book String String NonNegInt NonNegInt)
```
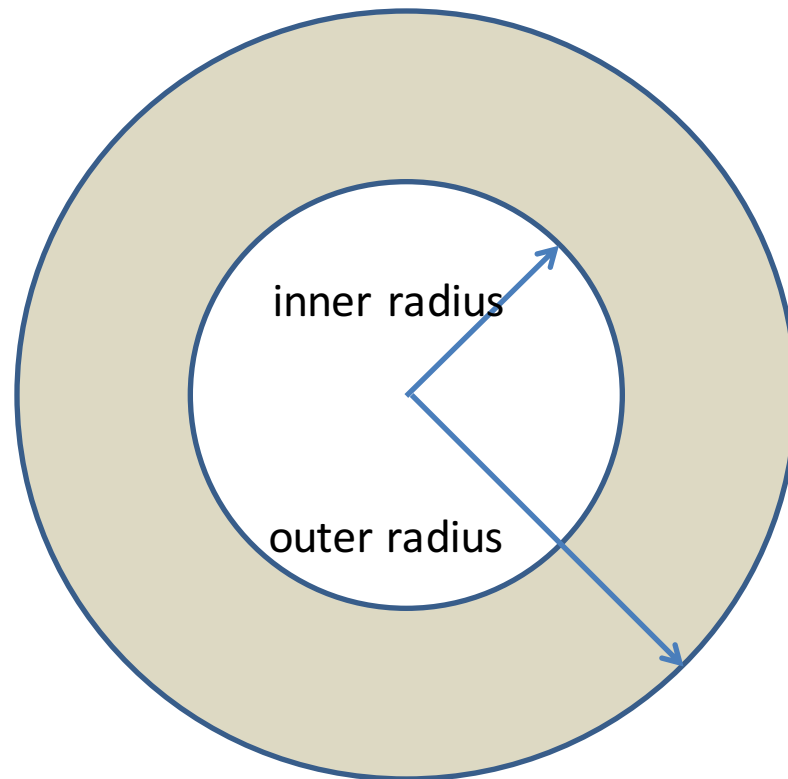
The kind of data in each field

Sometimes this format isn't enough

- Example: a ring

inner radius

outer radius

Constructor Template for a ring

- You could apply make-ring to any two numbers
- But only some combinations of the arguments make sense. We document this with the WHERE clause.

```
(define-struct ring (inner outer))
;; A Ring is a (make-ring PosNum PosNum)
;; WHERE: (< inner outer) is true.
```

Example of a constructor template: itemization data

- For itemization data, there are no struct definitions, and the constructor template just enumerates the possible values.

```
A TLState is one of
-- "red"
-- "yellow"
-- "green"
```

We use CamelCase for names of kinds of data

DDR Step 4. Interpretation

- Tells what piece of information each value of the data represents.

- The interpretation must show the interpretation of each alternative and the interpretation of each field.

- This usually refers to the structure definition and the constructor template, so these three pieces are written together.

struct definitions and constructor template for mixed data: example

```
(define-struct coffee (size type milk?))
(define-struct wine (vineyard year))
(define-struct tea (size type))
```

The structure definitions

```
;; A BarOrder is one of
;; -- (make-coffee Size Type Boolean)
;; -- (make-wine Vineyard Year)
;; -- (make-tea Size String)
```

The constructor template

Data Definition for mixed data: example

```
(define-struct coffee (size type milk?))
(define-struct wine (vineyard year))
(define-struct tea (size type))
```

The structure definitions

```
;; A BarOrder is one of
;; -- (make-coffee Size Type Boolean)
;;   INTERP:
;;     size is the size of cup desired
;;     type is the origin of the coffee
;;     milk? tells whether milk is desired.
;; -- (make-wine Vineyard Year)
;;   INTERP:
;;     vineyard is the origin of the grapes
;;     year is the year of harvest
;; -- (make-tea Size String)
;;   INTERP:
;;     size is the size of cup desired
;;     type is the type of tea (as a string)
```

Presumably Size and Type are data types defined elsewhere.

Here it's clear what the alternatives mean, so all we need to provide is the interpretation of each field in each alternative.

Presumably Vineyard is also a data type defined elsewhere.

Another example

```
(define-struct book (author title on-hand price))

;; A Book is a
;;   (make-book String String NonNegInt NonNegInt)
;; Interpretation:
;;    author is the author's name
;;    title is the title
;;    on-hand is the number of copies on hand
;;    price is the price in USD*100
;;     e.g. US$7.95 => 795
```

The structure definition

The constructor template

The interpretation of each field

Question: Are these interpretations sufficiently detailed to be useful?

Another example

- Do we need an interpretation?

```
A TLState is one of
-- "red"
-- "yellow"
-- "green"
```

- NO: common sense is good enough

Another example

- Do we need an interpretation?

```
A TLState is one of
-- 217
-- 126
-- 43
```

- YES: the reader is unlikely to guess that 217 denotes green, 126 denotes yellow, and 43 denotes red.

Not all integers are created equal

The interpretation tells you the meaning of each number. It also tells you that you shouldn't be adding these integers!

Summary

- You should now be able to write struct definitions, constructor templates, and interpretations for mixed, compound, and itemization data.

Next Steps

- If you have questions about this lesson, ask them on the Discussion Board
- Go on to the next lesson