The Design Process for CS2135

An example over list of circles

Create a data definition for a list of circles

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

[Can you make examples of this data?]

Create a data definition for a list of circles (with examples)

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

```
- empty
- (cons (make-circle (make-posn 0 0) 15)
        (cons (make-circle (make-posn –5 3) 40)
              empty))
```

Create a template for list-of-circles

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

Remember, a template captures the part of a program that we get "for free" from the structure of the data.

A template addresses the *input*, not the *output*, of a function

[Try writing the template]

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

First, give your template a name based on the name of the data

---

```
(define (locirc-func alocirc)
   … )
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

Next, exploit the cases in the data definition

```
(define (locirc-func alocirc)
   … )
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

Next, exploit the cases in the data definition

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
             [(cons? alocirc) …]))
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

Next, extract the components of the data in each case

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
            [(cons? alocirc) …]))
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

Next, extract the components of the data in each case

```
(define (locirc-func alocirc)
  (cond [(empty? alocirc) ...]
        [(cons? alocirc)
          ... (first alocirc) ...
            (rest alocirc) ... ]))
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

Finally, account for the arrows by calling functions for the target data definitions

---

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
            [(cons? alocirc)
              … (first alocirc) …
                 (rest alocirc) … ]))
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

Finally, account for the arrows by calling functions for the target data definitions

---

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
            [(cons? alocirc)
              … (circle-func (first alocirc)) …
                  (locirc-func (rest alocirc)) … ]))
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

But where is circle-func?

We need a template for it …

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
            [(cons? alocirc)
              … (circle-func (first alocirc)) …
                 (locirc-func (rest alocirc)) … ]))
```

When you create a template, write one template function for each data definition

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

Add a template for circles

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
            [(cons? alocirc) … (circle-func (first alocirc)) …
                                         (locirc-func (rest alocirc)) … ]))
(define (circle-func a-circ)
   …)
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

What goes into that template?

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
              [(cons? alocirc) … (circle-func (first alocirc)) …
                                            (locirc-func (rest alocirc)) … ]))
(define (circle-func a-circ)
   …)
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

There are no cases, so we move directly to extracting the components of the data

---

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
         [(cons? alocirc) … (circle-func (first alocirc)) …
                            (locirc-func (rest alocirc)) … ]))
(define (circle-func a-circ)
   (circle-center a-circ) …
   (circle-radius a-circ) …)
```

Creating a template list-of-circles: steps

```
;; A circle is a (make-circle posn number)
(define-struct circle (center radius))

;; A list-of-circles is either
;;    - empty, or
;;    - (cons circle list-of-circles)
```

There are no arrows from the circle defn to other defns, so the template is done

---

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
            [(cons? alocirc) … (circle-func (first alocirc)) …
                                        (locirc-func (rest alocirc)) … ]))
(define (circle-func a-circ)
   (circle-center a-circ) …
   (circle-radius a-circ) …)
```

Summary: Constructing the Template for a Data Definition

- Name the template function
- If the data defn has cases, add a cond with one clause per case
- For each case, use selectors (incl. first, rest) to extract the components of the datum
- Capture every arrow with a function call (this introduces the recursion)
- This may require additional template functions if multiple data definitions interact

From Templates to Functions

Starting from the template, write a function circle-areas that consumes a list of circles and produces a list of their areas

```
(define (locirc-func alocirc)
   (cond [(empty? alocirc) …]
              [(cons? alocirc) … (circle-func (first alocirc)) …
                                         (locirc-func (rest alocirc)) … ]))

(define (circle-func a-circ)
   (circle-center a-circ) …
   (circle-radius a-circ) …)
```

From Templates to Functions

First, write examples of how the new function should work.

Use the data examples developed with the data defn.

```
- (circle-areas empty) = empty
- (circle-areas
     (cons (make-circle (make-posn 0 0) 15)
              (cons (make-circle (make-posn –5 3) 40)
                        empty)))
   = (cons 706.5 (cons 5024 empty))
```

From Templates to Functions

Next, rename the template function and add contract/purpose

```
;; circle-areas : list-of-circle → list-of-num
;; produces list of areas of the circles in the list
(define (circle-areas alocirc)
   (cond [(empty? alocirc) …]
              [(cons? alocirc) … (circle-func (first alocirc)) …
                                        (circle-areas (rest alocirc)) … ]))

(define (circle-func a-circ)
   (circle-center a-circ) …
   (circle-radius a-circ) …)
```

From Templates to Functions

Next, use examples to fill in the …; Start with the base case

```
;; circle-areas : list-of-circle → list-of-num
;; produces list of areas of the circles in the list
(define (circle-areas alocirc)
   (cond [(empty? alocirc) empty]
              [(cons? alocirc) … (circle-func (first alocirc)) …
                                              (circle-areas (rest alocirc)) … ]))

(define (circle-func a-circ)
   (circle-center a-circ) …
   (circle-radius a-circ) …)
```

From Templates to Functions

For the recursive case, ask what each piece should give you

```
;; circle-areas : list-of-circle → list-of-num
;; produces list of areas of the circles in the list
(define (circle-areas alocirc)
   (cond [(empty? alocirc) empty]
         [(cons? alocirc) … (circle-func (first alocirc)) …
                             (circle-areas (rest alocirc)) … ]))

(define (circle-func a-circ)
   (circle-center a-circ) …
   (circle-radius a-circ) …)
```

need area of first circle

The list of areas of the circles in the rest of the list

From Templates to Functions

This tells you what circle-func needs to do – rename it and add contract/purpose (but finish it later)

```
;; circle-areas : list-of-circle → list-of-num
;; produces list of areas of the circles in the list
(define (circle-areas alocirc)
   (cond [(empty? alocirc) empty]
         [(cons? alocirc) … (area (first alocirc)) …
                            (circle-areas (rest alocirc)) … ]))
```

need area of first circle

The list of areas of the circles in the rest of the list

```
;; area : circle → number
;; calculates the area of a circle
(define (area a-circ)
   (circle-center a-circ) …
   (circle-radius a-circ) …)
```

From Templates to Functions

Determine how to combine the results on first and rest. Our output must be a list, so we probably want cons

```
;; circle-areas : list-of-circle → list-of-num
;; produces list of areas of the circles in the list
(define (circle-areas alocirc)
   (cond [(empty? alocirc) empty]
         [(cons? alocirc) (cons (area (first alocirc))
                                (circle-areas (rest alocirc)))]))
```

need area of first circle

The list of areas of the circles in the rest of the list

```
;; area : circle → number
;; calculates the area of a circle
(define (area a-circ)
   (circle-center a-circ) …
   (circle-radius a-circ) …)
```

From Templates to Functions

Circle-areas is done, so now go finish area

```
;; circle-areas : list-of-circle → list-of-num
;; produces list of areas of the circles in the list
(define (circle-areas alocirc)
   (cond [(empty? alocirc) empty]
         [(cons? alocirc) (cons (area (first alocirc))
                                (circle-areas (rest alocirc)))]))

;; area : circle → number
;; calculates the area of a circle
(define (area a-circ)
   (* pi (square (circle-radius a-circ))))
```

Summary of Design Process

- Develop data definitions and examples of the data
- Develop the template(s) for the data defns
- Once given a function to write, write examples of the expected output of the function
- Edit the template to complete the function definition
- Test the function with your examples