What is a house?

It depends on what you want to do with it.

Let’s suppose we want to draw it.

- Draw a rectangle.
- Put an triangle on top of it.

What information do we need?

- The location, width, and height of the rectangle.

We can compute the location, width, and height of the triangle from the description of the rectangle.
House data type, version 1

The house data type

A house is a structure with two fields, called “body” and “roof”.

The body is a rectangle.
The roof is a triangle.

Defining the house data type

Define a new data structure with “body” and “roof” fields:

(define-struct house (body roof))

Scheme automatically defines the following procedures:

(define make-house (lambda (body roof) ...))
(define house-body (lambda (house) ...))
(define house-roof (lambda (house) ...))
(define house? (lambda (thing) ...))

Public constructor function

Scheme defined make-house, but it doesn’t know how to make the appropriate triangle and rectangle!

(define build-house
  (lambda (left bottom width height)
    (make-house (build-rectangle left bottom width height)
                (build-triangle (+ left (/ width 2))
                                (+ bottom height)
                                width
                                height))))
(define lathrop-house (build-house 10 5 16 7))

Additional selector functions

(define house-left (lambda (house) (rectangle-left (house-body house))))
(define house-bottom (lambda (house) (rectangle-bottom (house-body house))))
(define house-width (lambda (house) (rectangle-width (house-body house))))
(define house-height (lambda (house) (rectangle-height (house-body house))))

Since left, width, and height are not stored explicitly in the house structure, they must be computed “on the fly” from the explicitly stored quantities.

Drawing a house

(define display-house (lambda (house)
  (display-rectangle (house-body house))
  (display-triangle (house-roof house))))

Notice that the programmer doesn’t need to think about how the structure is represented inside the computer. She only needs to remember the names of the selector functions.

House data type, version 2
**The house data type**

A **house** is a structure with four **fields**, called “left”, “bottom”, “width”, and “height”.

They define the size, shape, and position of the body of the house.

They indirectly determine the size, shape, and position of the roof of the house.

---

**Defining the house data type**

Define a new data structure with “left”, “bottom”, “width”, and “height” fields:

```
(define-struct house (left bottom width height))
```

Scheme automatically defines the following procedures:

```
(define make-house (lambda (l b w h) ...))
(define house-left (lambda (house) ...))
(define house-bottom (lambda (house) ...))
(define house-width (lambda (house) ...))
(define house-height (lambda (house) ...))
(define house? (lambda (thing) ...))
```

---

**Public constructor function**

```
(define build-house
 (lambda (left bottom width height)
  (make-house left bottom width height)))
```

```
(define lathrop-house
  (build-house 10 5 16 7))
```
Additional selector functions

```scheme
(define house-body
  (lambda (house)
    (build-rectangle (house-left house) (house-bottom house) (house-width house) (house-height house)))))

(define house-roof
  (lambda (house)
    (build-triangle (+ (house-left house) (/ (house-width house) 2)) (+ (house-bottom house) (house-height house)) (house-width house) (house-height house))))

Since body and roof are not stored explicitly in the house structure, they must be computed “on the fly” from the explicitly stored quantities.

Drawing a house

```scheme
(define display-house
  (lambda (house)
    (draw-rectangle (house-body house))
    (draw-triangle (house-roof house))))

Notice that the rectangle drawing procedure doesn't need to change at all.

Comparing the implementations

In the first implementation, the body and the roof are constructed by the `build-house` constructor function when the house itself is constructed.

In the second implementation, the body and the roof are not constructed by `build-house`. They are constructed later on when they are needed by the `house-body` and `house-roof` selector functions.
The two implementations have the same public interface

- build-house
- house-left
- house-bottom
- house-width
- house-height
- house-body
- house-roof

Store vs compute tradeoff

Store:
- Compute the information in advance.
- Look up the information when it’s needed.

Compute:
- Don’t compute the information in advance.
- Compute the information from when it’s needed.

Representing towers
The **tower** data type

A **tower** is a structure with four *fields*, called “base” and “roof”.

The base is a list of rectangles.

The roof is a triangle.

---

Defining the **tower** data type

Define a new data structure with “base” and “roof” fields:

```
(define-struct tower (base roof))
```

Scheme automatically defines the following procedures:

```
(define make-tower (lambda (body roof) ...))
(define tower-base (lambda (tower) ...))
(define tower-roof (lambda (tower) ...))
(define tower? (lambda (thing) ...))
```

---

Public constructor function

```
(define build-tower
 (lambda (n left bottom width height)
  (make-tower
   (build-tower-base n left bottom width height)
   (build-triangle (+ left (/ width 2))
    (+ bottom (* n height))
    width
    height))))
```

---

Building a tower base of $n$ blocks

If $n = 0$:

Do nothing

Otherwise:

1. Build one block and put it on the ground.
2. Build a tower base of $n-1$ blocks and put it on top of the first block.
(define build-tower-base
  (lambda (n left bottom width height)
    (if (= n 0)
        '()
        (cons (build-rectangle
                left
                bottom
                width
                height)
              (build-tower-base
                (- n 1)
                left
                (+ bottom height)
                width
                height)))))

(define jewett
  (build-tower 3 10 16 7))

Representing staircases

Building a staircase

Construct a stack of rectangles.
Each one is stacked on top of the previous one.
Each one is narrower than the previous one.
Each one is displaced with respect to the previous one.
A staircase is a very simple structure. It just has one field, which is the list of steps:

\[(\text{define-struct staircase} \ (\text{steps}))\]

Why don’t we have a separate field for each step?

**Public constructor function**

\[(\text{define build-staircase} \ (\lambda (n \ \text{left} \ \text{bottom} \ \text{width} \ \text{height})
\ (\text{make-staircase}
\ (\text{build-staircase-steps} \ n \ \text{left}
\ \text{bottom} \ \text{width} \ \text{height}))))\]

**Building an n-step staircase**

If \(n = 0\):

Do nothing

Otherwise:

1. Build one step and put it on the ground
2. Build an \(n-1\)-step staircase and put it on the first step
(define build-staircase-steps
  (lambda (n left bottom width height)
    (if (= n 0)
        '()
        (cons (build-rectangle left bottom (* n width) height)
              (build-staircase-steps (- n 1)
                left
                (+ bottom height)
                width
                height)))))

(displaying towers and staircases

(define display-object
  (lambda (x)
    (cond ((triangle? x)
           (display-triangle x))
          ((rectangle? x)
           (display-rectangle x))
          ((house? x)
           (display-house x))
          ((tower? x)
           (display-tower x))
          ((staircase? x)
           (display-staircase x))
          (else #f)))))

(define display-object-list
  (lambda (l)
    (cond ((null? l)
            #t)
          (else
           (display-object (first l))
           (display-object-list (rest l))))))
(define display-tower
  (lambda (tower)
    (display-triangle (tower-roof tower))
    (display-object-list
     (tower-base tower))))

(define display-staircase
  (lambda (staircase)
    (display-object-list
     (staircase-steps staircase))))

Acknowledgments

This lecture incorporates material from:

Tom Ellman