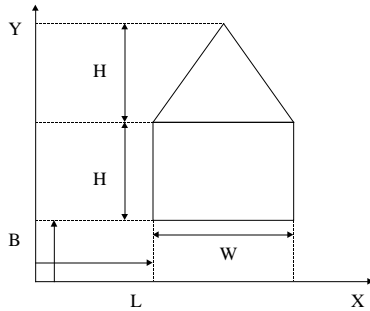


Computer Science I

Professor Tom Ellman
Lecture 14

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.
 - Compute the location, width and height of the triangle from the description of the rectangle.



How Shall We Implement a House? Implementation #1

- The symbol HOUSE.
- The body of the house (rectangle).
- The roof of the house (triangle).

Constructor Functions

```
(define make-house
  (lambda (body roof) (list 'house body roof)))

(define build-house
  (lambda (left bottom width height)
    (make-house (make-rectangle left bottom height width)
                (make-triangle (+ left (/ width 2))
                              (+ bottom height)
                              width
                              height))))
```

Comments on the Constructors

- The function MAKE-HOUSE implements the decision to represent a house as a list with a symbol, a rectangle and a triangle.
- The function BUILD-HOUSE provides a convenient way to create a house data structure from the L, B, W and H parameters.

Selector Functions

```
(define house-body (lambda (house) (cadr house)))  
(define house-roof (lambda (house) (caddr house)))
```

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))))  
  
(define house?  
  (lambda (house) (and (pair? house)  
                       (equal? (car house) 'house))))
```

Comments on the Selectors

- Since the body and roof are created in advance, HOUSE-BODY and HOUSE-ROOF simply get them off of the list.
- The parameters L, B, W and H are obtained by first using HOUSE-BODY to get a rectangle, and then using the rectangle selector functions.

Drawing a House

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

Since the house is implemented as a rectangle and a triangle, we can use the previously written drawing functions for these types of objects.

How Shall We Implement a House? Implementation #2

- The symbol HOUSE.
- The four parameters L, B, W and H.

Constructor Functions

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

Comments on the Constructors

- The function MAKE-HOUSE implements the decision to represent a house as a list with a symbol and the L, B, W, H parameters.
- The function BUILD-HOUSE just calls MAKE-HOUSE with the same parameters.

Why do we need BUILD-HOUSE ?

Selector Functions

```
(define house-left (lambda (house) (cadr house)))  
(define house-bottom (lambda (house) (caddr house)))  
(define house-width (lambda (house) (caddr house)))  
(define house-height (lambda (house) (car (cddddr house))))
```

Selector Functions

```
(define house-body (lambda (house)  
  (make-rectangle (house-left house)  
                 (house-bottom house)  
                 (house-width house)  
                 (house-height house))))  
  
(define house-roof (lambda (house)  
  (make-triangle (+ (house-left house)  
                  (/ (house-width house) 2))  
                (+ (house-bottom house)  
                  (house-height house))  
                (house-width house)  
                (house-height house))))
```

Comments on the Selectors

- Since the parameters L, B, W and H are contained in the HOUSE data structure, the selector functions simply retrieve them from the list.
- The body and roof are not created in advance, so HOUSE-BODY and HOUSE-ROOF must construct them.

Drawing a House

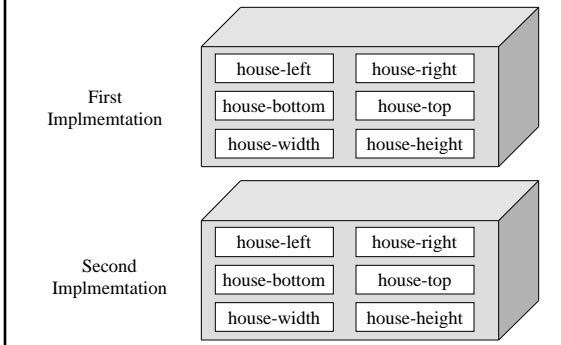
```
(define draw-house (lambda (house)  
  (begin (draw-rectangle (house-body house))  
        (draw-triangle (house-roof house)))))
```

Even though the house is no longer implemented as a list with a rectangle and a triangle, we have selector functions that create these data. We can still use the same drawing function.

Comparing the Two Implementations

- In the first implementation the body and the roof are constructed by the MAKE-HOUSE constructor function when the house itself is constructed.
- In the second implementation, the body and the roof are not constructed by MAKE-HOUSE. They are constructed later on when they are needed by the HOUSE-BODY and HOUSE-ROOF selector functions.

The Two Implementations are Identical when Accessed via Selector Functions



Store v. 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.

Comparing the Two Implementations

- We have two different implementations of the constructor and selector functions.
 - The function names are the same.
 - The inputs are the same.
 - The return values are the same.
- The two implementations have the same behavior, even though the function definitions are different.

Hiding Information

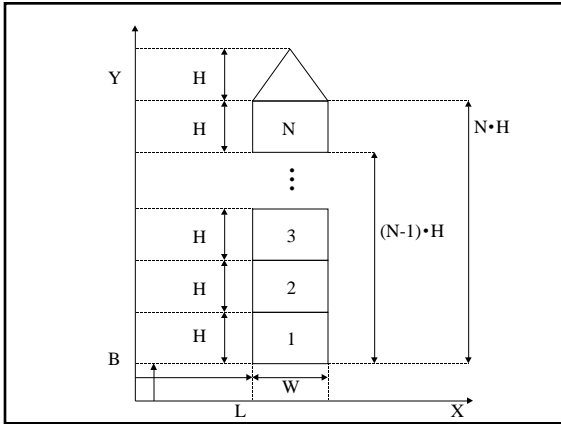
- The constructor and selector functions are collectively called "access functions".
- The access functions define an "interface" to the house data structure.
- They are said to "encapsulate the data" in a "black box".
- They "hide" the implementation of the house data type from the programmer who uses the access functions.

Why is Information Hiding Good?

- Suppose I write house access functions using the first implementation.
- Then I write the house drawing function.
- Suppose I later change my mind and rewrite the house access functions using the second implementation.
- I don't need to change the drawing function!

Building a Tower

- Construct a stack of rectangles with each one stacked on top of the the previous one.
- Put a triangle on top.



Constructor Functions

```
(define make-tower (lambda (base roof) (list 'tower base roof)))
```

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

Building a Tower Base of N Blocks

If $N = 0$

Then Do Nothing.

Otherwise 1. Build a Tower Base of $N-1$ Blocks.

2. Put Another Block on Top.

```
(define build-tower-base
  (lambda (n left bottom width height)
    (if (= n 0)
        '()
        (cons (make-rectangle left
                               (+ bottom (* (- n 1) height))
                               width
                               height)
              (build-tower-base (- n 1) left bottom width height))))
```

Selector Functions

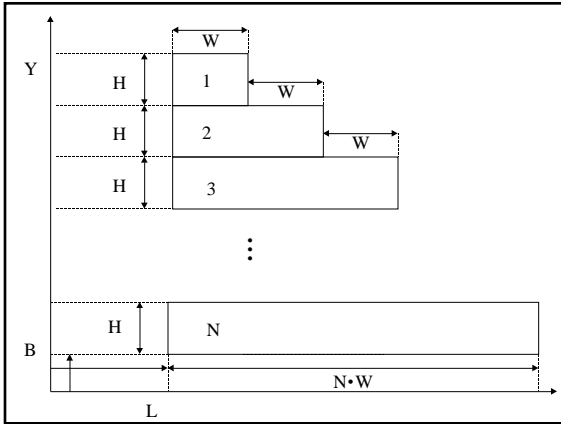
```
(define tower-base (lambda (tower) (cadr tower)))
```

```
(define tower-roof (lambda (tower) (caddr tower)))
```

```
(define tower?
  (lambda (tower) (and (pair? tower)
                       (equal? (car tower) 'tower))))
```

Building a Staircase

- Construct a stack of rectangles.
- Each one stacked on top of the previous one.
- Each one narrower than the previous one.
- Each one displaced w.r.t. the previous one.



Constructor Functions

```
(define make-staircase (lambda (steps) (list 'staircase steps)))
```

```
(define build-staircase
  (lambda (n left bottom width height)
    (make-staircase (build-staircase-steps n
                                           left
                                           bottom
                                           width
                                           height))))
```

Building N Steps

If $N = 0$

Then Do Nothing.

Otherwise 1. Build $N-1$ Steps.

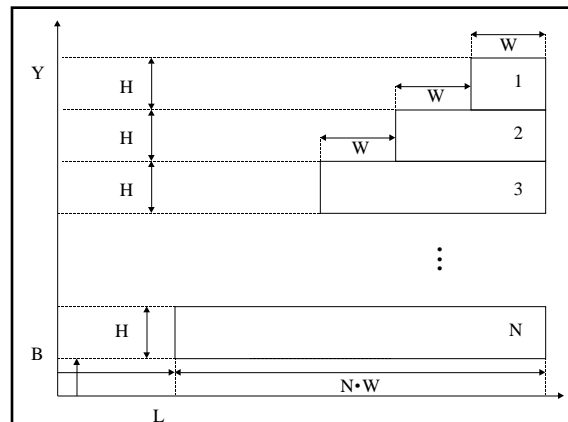
2. Put Another Step Underneath.

```
(define build-staircase-steps
  (lambda (n left bottom width height)
    (if (= n 0)
        '()
        (cons (make-rectangle left bottom (* n width) height)
              (build-staircase-steps (- n 1)
                                     left
                                     (+ bottom height)
                                     width
                                     height)))))
```

Selector Functions

```
(define staircase-steps (lambda (staircase) (cadr staircase)))
```

```
(define staircase?
  (lambda (staircase) (and (pair? staircase)
                            (equal? (car staircase) 'staircase))))
```



```
(define build-staircase-steps
  (lambda (n left bottom width height)
    (if (= n 0)
        '()
        (cons (make-rectangle left bottom (* n width) height)
              (build-staircase-steps (- n 1)
                                     (+ left width)
                                     (+ bottom height)
                                     width
                                     height))))))
```