Exercise 2.3.

Implement a representation for rectangles in a plane. (Hint: You may want to make use of exercise 2.2.) In terms of your constructors and selectors, create procedures that compute the perimeter and the area of a given rectangle. Now implement a different representation for rectangles. Can you design your system with suitable abstraction barriers, so that the same perimeter and area procedures will work using either representation?

Answer.

Before setting out to implement a rectangle, let's ask ourselves a question: what is the most general practice for computing the perimeter and the area of a rectangle? Well, that would be:

$$\begin{aligned} C &= 2\,(w+h) \\ S &= w\,h \end{aligned}$$

where the w and h here stand for the width and heigh of a rectangle with respect.

By wishful thinking, to evaluate the perimeter and area of a particular rectangle, all that we need to know is its width and heigh. Suppose we have a way of extracting the width and heigh of a given rectangle, say, the procedures `width-rec` and `heigh-rec`. We can express the idea for evaluating the perimeter and the area of a rectangle with these two procedures, despite of their implementation:

```
(define (peri-rec r)
  (* (+ (width-rec r) (heigh-rec r)))
     2))

(define (area-rec r)
  (* (width-rec r) (heigh-rec r)))
```

The Usual Representation of a Rectangle

In order to implement the delayed procedure `width-rec` and `heigh-rec` above, we have to consider the concrete representation of a rectangle. A usual strategy would be by two adjcent segments, for example m and l, as is shown in Figure 1. Thus, we can readily express this idea in Lisp as:

```
(define (make-rec m l) (cons m l))

(define (m-rec r) (car r))

(define (l-rec r) (cdr r))
```

We can write the procedure `width-rec` and `heigh-rec` in terms of this representation with the help of `length`, which is a procedure to measure the length of a segment.

```
(define (width-rec r) (length (m-rec r)))

(define (heigh-rec r) (length (l-rec r)))
```

The function of procedure `length` here is to measure the length of a line segment and is defined to be:

```
(define (length s)
  (let ((dx (- (x-point (end-segment s))
               (x-point (start-segment s))))
        (dy (- (y-point (end-segment s))
               (y-point (end-segment s)))))
    (define (pythagoras x y)
      (sqrt (+ (square x) (square y))))
    (pythagoras dx dy)))
```

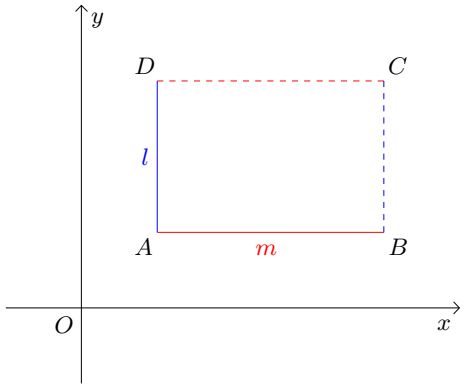

Figure 1. Representing a Rectangle by Two Adjcent Line Segments

An Alternative Way for Representing a Rectangle

Besides the usual practice, there is another approach to representating a rectangle. Figure 2 shows such a way geometrically by using two parallel line segments, say, m and n. This reveals a different way to to representing a rectangle in Lisp:

```
(define (make-rec m n) (cons m n))

(define (m-rec r) (car r))

(define (n-rec r) (cdr r))
```

Using this representation, the procedure `width-rec` and `heigh-rec` should be implemented in a different way:

```
(define (width-rec r) (length (m-rec r)))

(define (heigh-rec r)
  (let ((h (make-segment (start-segment (m-rec r))
                         (start-segment (n-rec r)))))
    (length h)))
```

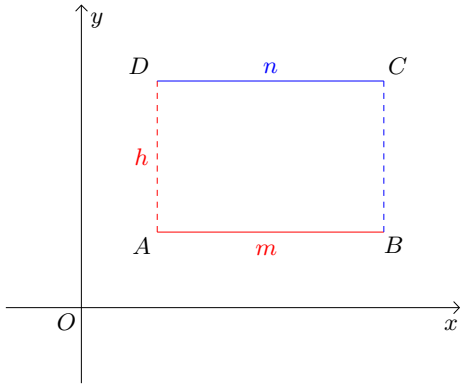

Figure 2. Representing a Rectangle by Two Parallel Line Segments