Peter Henderson’s Picture Language
Structure and Interpretation of Computer Programs
Spring Term, 2004
The Picture Language

*Invented by* Peter Henderson (Oxford/Southhampton)

*Motivation:*

VLSI circuit design--
eliminate repetitive wire layout
succinct description via a good programming language
   (idea of embedded languages)
Escher diagrams (pretty pictures)

*Our interest:* higher-order procedures.

(We try to minimize vector calculations -- this is not (!!) a course in linear algebra...)

Try and read this code and guess what it does
(like reading a foreign language... pretend you understand...)
Note: I forget and relearn this code, every year I teach the course
   ("How's that, again? ...")

Hint: a painter is something that draws a picture.
(*How? What?* Suspend your ignorance for a moment...)
A painter is something that draws a picture...

(define wave2 (beside wave (flip-vert wave)))
(define wave4 (below wave2 wave2))
A painter is something that draws a picture...

(define (right-split painter n)
  (if (= n 0)
      painter
      (let ((smaller (right-split painter (- n 1))))
        (beside painter (below smaller smaller))))

(define (corner-split painter n)
  (if (= n 0)
      painter
      (let ((up (up-split painter (- n 1)))
        (right (right-split painter (- n 1))))
        (let ((top-left (beside up up))
          (bottom-right (below right right))
          (corner (corner-split painter (- n 1))))
          (beside (below painter top-left)
                  (below bottom-right corner))))))
A painter is something that draws a picture...

(define wave2 (beside wave (flip-vert wave)))
(define wave4 (below wave2 wave2))

(define (flipped-pairs painter)
  (let ((painter2 (beside painter (flip-vert painter))))
    (below painter2 painter2)))

(define wave4 (flipped-pairs wave))

(define (right-split painter n)
  (if (= n 0)
    painter
    (let ((smaller (right-split painter (- n 1))))
      (beside painter (below smaller smaller)))))

(define (corner-split painter n)
  (if (= n 0)
    painter
    (let ((up (up-split painter (- n 1)))
      (right (right-split painter (- n 1))))
      (let ((top-left (beside up up))
        (bottom-right (below right right))
        (corner (corner-split painter (- n 1))))
        (beside (below painter top-left) (below bottom-right corner)))))

(define (square-limit painter n)
  (let ((quarter (corner-split painter n)))
    (let ((half (beside (flip-horiz quarter) quarter)))
      (below (flip-vert half) half))))
Transforming painters

(define (rotate90 painter)
  (transform-painter painter
    (make-vect 1.0 0.0)
    (make-vect 1.0 1.0)
    (make-vect 0.0 0.0)))

(define (squash-inwards painter)
  (transform-painter painter
    (make-vect 0.0 0.0)
    (make-vect 0.65 0.35)
    (make-vect 0.35 0.65)))

(define (beside painter1 painter2)
  (let ((split-point (make-vect 0.5 0.0)))
    (let ((paint-left
            (transform-painter painter1
              (make-vect 0.0 0.0)
              split-point
              (make-vect 0.0 1.0)))
          (paint-right
            (transform-painter painter2
              split-point
              (make-vect 1.0 0.0)
              (make-vect 0.5 1.0))))
      (lambda (frame)
        (paint-left frame)
        (paint-right frame))))
Frames

Origin(Frame) + x*Edge1(Frame) + y*Edge2(Frame)

(define (frame-coord-map frame)
  (lambda (v)
    (add-vect
      (origin-frame frame)
      (add-vect (scale-vect (xcor-vect v)
                      (edge1-frame frame))
                (scale-vect (ycor-vect v)
                            (edge2-frame frame))))

  ((frame-coord-map a-frame) (make-vect 0 0))

  (origin-frame a-frame))
Painters

(define (transform-painter painter origin corner1 corner2)
  (lambda (frame)
    (let ((m (frame-coord-map frame)))
      (let ((new-origin (m origin)))
        (painter
          (make-frame new-origin
            (sub-vect (m corner1) new-origin)
            (sub-vect (m corner2) new-origin)))))))

(define (flip-vert painter)
  (transform-painter painter
    (make-vect 0.0 1.0)    ; new origin
    (make-vect 1.0 1.0)    ; new end of edge1
    (make-vect 0.0 0.0)))  ; new end of edge2)

(define (shrink-to-upper-right painter)
  (transform-painter painter
    (make-vect 0.5 0.5)    ; new origin
    (make-vect 1.0 0.5)    ; new end of edge1
    (make-vect 0.5 1.0)))
Functionality

Paint\(\text{er} = \text{Frame} \rightarrow \text{Output} \quad [\text{draws a picture in a frame}]\)

\(\text{Frame} = \text{Vector} \times \text{Vector} \times \text{Vector}\)

\text{transform-painter: Painter} \times \text{Vector} \times \text{Vector} \times \text{Vector} \rightarrow \text{Painter}\)

\text{above, below, beside, rotate90: Painter} \rightarrow \text{Painter}