CS 173 Programming Languages Shriram K.

Scheme Tutorial Solutions

Fall 2003

Problem Set 1: Basic Scheme

1. Function to total the amount of change (pennies, nickels, dimes, quarters) in a bag:

```
;; sum-coins : number number number number → number
(define (sum-coins pen nick dime quart)
  (+ (∗ .01 pen)
     (∗ .05 nick)
     (∗ .1 dime)
     (∗ .25 quart)))
```

2. Function to compute the surface area of a cylinder:

```
;; area-cylinder : number number → number
(define (area-cylinder radius height)
  (+ (∗ 2 pi (sqr radius))
     (∗ 2 pi radius height)))
```

3. Surface area of a pipe computed as a single function:

```
;; area-pipe1 : number number number → number
(define (area-pipe1 inner-radius height thickness)
  (+ (∗ 2 pi height inner-radius)
     (∗ 2 pi height (+ thickness inner-radius))
     (∗ 2 (− (∗ pi (sqr (+ thickness inner-radius)))
             (∗ pi (sqr inner-radius))))))
```

Surface area of a pipe computed using helper functions:

```
;; area-pipe : number number number → number
;; to determine the area of a pipe with given inner radius, length, and
;; thickness
(define (area-pipe inner-radius height thickness)
  (+ (∗ height (circumference (+ inner-radius thickness)))
     (∗ height (circumference inner-radius))
     (∗ 2 (− (area-circle (+ inner-radius thickness))
             (area-circle inner-radius)))))
```

```
;; area-circle : number → number
;; determines the area of a circle with given radius
(define (area-circle r)
  (∗ pi r r))

;; circumference : number → number
;; determines the circumference of a circle with given radius
(define (circumference r)
  (∗ 2 pi r))
```

4. Function for computing tax:

```
;; tax : number → number
;; computes a flat income tax
(define (tax pay)
  (cond
    [(<= pay 240) 0]
    [(> pay 480) (∗ pay .28)]
    [else (∗ pay .15)]))
```

Functions for computing gross pay and net pay (based on gross pay):

```
;; gross-pay : number → number
;; computes the gross pay of a person making $12 an hour, based on the hours worked
(define (gross-pay hours)
  (∗ 12 hours))

;; net-pay : number → number
;; computes the net pay based on hours worked
(define (net-pay hours)
  (− (gross-pay hours) (tax (gross-pay hours))))
```

5. Functions to determine if a quadratic equation is degenerate or not. If it is not degenerate it then computes whether the solution has 2, 1 or 0 solutions.

```
;; discriminant : number number number → number
(define (discriminant a b c)
  (− (sqr b) (∗ 4 a c)))
```

```
;; what-kind? : number number number → symbol
;; determines if a quadratic equation is degenerate, or has none,
;; one, or two solutions
(define (what-kind? a b c)
  (cond
    [(= a 0) 'degenerate]
    [(> (discriminant a b c) 0) 'two]
    [(= (discriminant a b c) 0) 'one]
    [else 'none]))
```

6. Function to compute the difference in seconds between two points in time, using the datatype *time-point* to represent hours, minutes and seconds.

```
;; datatype to represent time in hours, minutes and seconds
(define-datatype time time?
  [time-point (hour number?) (min number?) (sec number?)])

;; in-seconds : time → number
;; converts hour, minute and second representation of time into seconds
(define (in-seconds t)
  (cases time t
    [time-point (h m s) (+ s (* m 60) (* h 60 60))]))

;; time-diff : time-point time-point → number
;; computes the difference (in time sec) between two time-point(s)
(define (time-diff t1 t2)
  (- (in-seconds t2) (in-seconds t1)))
```

7. Datatype for representing a 2D-point and a shape.

```
(define-datatype position position?
  [2d-point (x number?)
            (y number?)])

(define-datatype shape shape?
  [circle (center position?)
          (radius number?)]
  [square (top-left position?)
          (length number?)]
  [rect (top-left position?)
        (width number?)
        (height number?)])
```

8. Function for finding the area of a shape

```
;; area : shape → number
(define (area s)
  (cases shape s
    [square (tl l) (sqr l)]
    [rect (tl w h) (* w h)]
    [circle (c r) (* pi (sqr r))]))
```

9. Functions which take a shape and return a new shape. The new shape is a copy of the old shape translated by a value in the x direction

```
;; getx : position → number
(define (getx p)
  (cases position p
    [2d-point (x y) x]))

;; gety : position → number
(define (gety p)
  (cases position p
    [2d-point (x y) y]))

; translate-shape : shape number → shape
; translates a shape by a delta in the x direction
(define (translate-shape s delta)
  (cases shape s
    [square (tl l) (square (2d-point (+ delta (getx tl)) (gety tl)) l)]
    [rect (tl w h) (rect (2d-point (+ delta (getx tl)) (gety tl)) w h)]
    [circle (c r) (circle (2d-point (+ delta (getx tl) (gety tl))) r)]))
```

10. Functions to determine if a point is within a shape:

```
;; between? : number number number → boolean
;; determines if the first number is within the range of the second two.
(define (between? x l r)
  (and (>= x l) (<= x r)))

;; in-circle? : point number point → boolean
(define (in-circle? center radius pt)
  (<= (+ (sqr (- (getx pt) (getx center)))
         (sqr (- (gety pt) (gety center))))
      (sqr radius)))

;; in-square? : point number point → boolean
(define (in-square? tl l pt)
  (and (between? (getx pt)
                 (getx tl)
                 (+ (getx tl) l))
       (between? (gety pt)
                 (gety tl)
                 (+ (gety tl) l))))

;; in-rectangle?: point number number point → boolean
(define (in-rectangle? tl width height pt)
  (and (between? (getx pt)
                 (getx tl)
                 (+ (getx tl) width))
       (between? (gety pt)
                 (gety tl)
                 (+ (gety tl) height))))

;; in-shape? : shape point → boolean
(define (in-shape? s pt)
  (cases shape s
    [circle (c r) (in-circle? c r pt)]
    [square (tl l) (in-square? tl l pt)]
    [rect (tl w h) (in-rectangle? tl w h pt)]))
```