

7a: [HKK 7.44 with the last part changed] Consider the following two procedures. The procedure `last` selects the last element from a list, which must be nonempty. It uses `length` to find the length of the list.

```
(define last
  (lambda (lst)
    (if (= (length lst) 1)
        (car lst)
        (last (cdr lst)))))

(define length
  (lambda (lst)
    (if (null? lst)
        0
        (+ 1 (length (cdr lst))))))
```

As always, an explanation is appropriate for each of the following questions; when counting `cdrs` be sure to count those done by either procedure.

- (a) How many `cdrs` does `(length lst)` do when *lst* has *n* elements?
- (b) How many calls to `length` does `(last lst)` make when *lst* has *n* elements?
- (c) Express in Θ notation the total number of `cdrs` done by `(last lst)`, including `cdrs` done by `length`, again assuming that *lst* has *n* elements.
- (d) Write a new version of `last` that is more efficient, in the sense that the number of `cdrs` is improved by more than merely a constant factor, as expressed by Θ -notation. Express the number of `cdrs` performed by your improved version using Θ -notation.

7b: [HKK 7.46] Write a higher-order procedure `make-list-scaler` that takes a single number *scale* and returns a procedure that, when applied to a list *lst* of numbers, will return the list obtained by multiplying each element of *lst* by *scale*. Thus, you might have the following interaction:

```
(define scale-by-5 (make-list-scaler 5))

(scale-by-5 '(1 2 3 4))
(5 10 15 20)
```

7c: [HKK 7.50] Consider the following procedure (together with two sample calls):

```
(define repeat
  (lambda (num times)
    (if (= times 0)
        '()
        (cons num (repeat num (- times 1))))))

(repeat 3 2)
(3 3)

(repeat 17 5)
(17 17 17 17 17)
```

- (a) Explain why `repeat` generates a recursive process.
- (b) Write an iterative version of `repeat`.

7d: [HKK 7.49] Given a predicate that tests a single item, such as `positive?`, we can construct an “all are” version of it for testing a list; an example is a predicate that tests whether all elements of a list are positive. Define a procedure `all-are` that does this; that is, it should be possible to use it in ways like the following:

```
((all-are positive?) '(1 2 3 4))  
#t
```

```
((all-are even?) '(2 4 5 6 8))  
#f
```

- 7e:** [HKK 7.19] When our children started bringing home dozens of cheap plastic spiders, we asked them to restrict themselves to getting only one of each kind of prize. Write a procedure that is given the prize list and amount and computes the number of prize combinations that you can buy using exactly that amount and assuming that you can't get more than one of any particular prize. Naturally, with any subtle problem such as this, you should include carefully chosen test case(s); show that your program gives the *right* answer on your test cases by hand-computing the answer.
- 7f:** [HKK 7.20] Write another procedure that will determine the number of combinations you can buy using no more than a maximum amount of tickets while still insisting that you can have at most one of each kind of prize. As in 7.19, assume that only one of each prize is allowed. Include carefully chosen test case(s); show that your program gives the *right* answer on your test cases by hand-computing the answer.
- 7g:** Reread the syllabus, including all course policies. Make three comments concerning the syllabus. Each comment can be
- (a) something that surprised you about the syllabus
 - (b) something about the syllabus you recommend I change, or
 - (c) something about the syllabus you are especially fond of