CS61A Exam Prep 11

Summer 2021 final Q5a

Definition: Each element of the `fibonacci2` sequence is defined as twice the absolute value of the difference between the previous two elements. Assume that the 0th element of the `fibonacci2` sequence is 0, and the 1st element is 1.

Implement the function `fib2`, which takes in one parameter `n`, a non-negative integer, and returns the `n`th element of the `fibonacci2` sequence.

Reminder: Scheme has a built in procedure `abs` which returns the absolute value of the argument that is passed in.

```
                                 (-n 1)      (-n 2)
(define (fib2 n)
    (if (<= n 1) n
       ; (a)
       (* 2 (abs (- (fib2 (-n 1)) (fib2 (-n2)))))))
         ; (b)  (c)  (d)   (e)          (f)
(expect (fib2 0) 0)
(expect (fib2 1) 1)
(expect (fib2 2) 2)
(expect (fib2 3) 2)
(expect (fib2 4) 0)
(expect (fib2 5) 4)
```

Summer 2021 final Q5b

Definition: The *countdown sequence* of a number `n` is the sequence starting at `n` and descending to 0. For example, the *countdown sequence* of 3 is `3 2 1 0`.

Implement a function `countdowns` which takes in a scheme list `lst` of non-negative integers and returns a list which is the concatenation of the *countdown sequences* of each element in `lst`.

```
(define (countdowns lst)
   (cond ((null? lst) ___nil______)   '()
                          ; (k)
        [((> (car lst)__ 0)] (cons (car lst)
              ; (l))         (cons (- (car lst) 1)
             (countdowns ___(cdr lst))__)))
                              ; (m)
        (else (cons 0 (countdown (cdr lst))))))
                              ; (o)
(expect (countdowns '(3)) (3 2 1 0))
(expect (countdowns '(2 0 3)) (2 1 0 0 3 2 1 0))
(expect (countdowns '()) ())
```

(2 0 3)

(1 0 3)

(0 0 3)

(0 3)

(3)

Spring 2019 Final Q7a

(a) (6 pt) The `count-evens` procedure takes a list of integers and returns the number of elements that are even. Rewrite `count-evens` as a tail-call optimized procedure by filling in the blanks below.

```
(define (count-evens ints)
    (cond ((null? ints) 0)
          ((even? (car ints)) (+ 1 (count-evens (cdr ints))))
          (else (count-evens (cdr ints)))))
```

```
(define (count-evens-tail ints)

    (define (helper ints total)

        (cond ((null? ints) total )

              ((even? (car ints)) (helper (cdr ints) (+total 1)))

              (else (helper (cdr ints) total)) ))

    (helper ints 0))
```

Spring 2017 final Q8

Fill in the Scheme **pairs** function so that (pairs L), where L is a list, produces a list of lists, where each of these lists contains a pair of elements from L. The function must be tail-recursive. You need not define (or use) the reverse function.

```
scm> (pairs '(1 2 3 4))
((1 2) (3 4))
scm> (pairs '(1 2 3 4 5))  ; Odd element at end put in singleton list.
((1 2) (3 4) (5))
scm> (pairs '())
()
```

(pairs '(2))
((2))

```
(define (reverse P)
  """Returns the reverse of list P. This function is tail-recursive"""
  ;;; Implementation not shown
)

(define (pairs L)

  (define (accum-pairs lst result)

    (cond ((null? lst)                                   result)

          ((null? (cdr lst))

           (cons (list (car lst)) result))

          (else (accum-pairs (cdr (cdr lst))

                             (cons (list (car lst) (car (cdr lst)))

                             result))))

  (reverse (accum-pairs L nil))
)
```

go.cs61a.org/melanie-feedback