[SPOILERS] Practice Problems for Section 5 -- Tests

Subscribe for email updates. PINNED

No tags yet. + Add Tag

Pavel Lepin COMMUNITY TA · 14 days ago

```
#lang racket
;; Based on the provided test file for HW4 in proglang.

;; INSTRUCTIONS:
;; Put into the same folder as the file with your practice problem solutions.
;; Make sure it starts with:
;;     #lang racket
;;     (provide (all-defined-out))
(require "Put the name of the file with your solutions here")
(require rackunit)
(require rackunit/text-ui)

;; Some helpers for testing streams ;;

(define (stream-ith stream n)
  (define pair (stream))
  (if (zero? n)
      (car pair)
      (stream-ith (cdr pair) (sub1 n))))

(define (test-nats)
  (define (nums n)
    (lambda () (cons n (nums (add1 n)))))
  ((nums 1)))

(define tests
  (test-suite
   "Model tests for section 5 practice problems"

   ;;; Warm Up ;;;
   ;; Knights Who Say 'Ni' ;;
   (check-equal? (factorial 4) 24 "factorial test #1")
   (check-equal? (factorial 0) 1 "factorial test #2")
   (check-equal? (factorial 1) 1 "factorial test #3")
   (check-equal? (factorial 5) 120 "factorial test #4")
   (check-equal? (factorial 7) 5040 "factorial test #5")

   ;; Another Shrubbery! ;;

   ;; NOTE: These tests do not verify that your solution is tail-recursive. ;;
   ;; You should do that yourself! ;;

   (check-equal? (tr-factorial 4) 24 "tr-factorial test #1")
   (check-equal? (tr-factorial 0) 1 "tr-factorial test #2")
   (check-equal? (tr-factorial 1) 1 "tr-factorial test #3")
   (check-equal? (tr-factorial 5) 120 "tr-factorial test #4")
   (check-equal? (tr-factorial 7) 5040 "tr-factorial test #5")

   ;; Palindromic Addition ;;
   (check-equal? (palindromic (list 1 2 4 8)) (list 9 6 6 9) "palindromic test #1")
   (check-equal? (palindromic null) null "palindromic test #2")
   (check-equal? (palindromic (list 25 13 8 9 22)) (list 47 22 16 22 47) "palindromic
test #3")

   ;;; Streams ;;;

   ;; Undecided ;;
   (check-equal? (stream-ith undecided 0) #t "undecided test #1")
   (check-equal? (stream-ith undecided 1) #f "undecided test #2")
   (check-equal? (stream-ith undecided 2) #t "undecided test #3")
   (check-equal? (stream-ith undecided 5) #f "undecided test #4")
   (check-equal? (stream-ith undecided 118) #t "undecided test #5")

   ;; Fibonacci ;;
   (check-equal? (stream-ith fibonacci 0) 0 "fibonacci test #1")
   (check-equal? (stream-ith fibonacci 1) 1 "fibonacci test #2")
   (check-equal? (stream-ith fibonacci 2) 1 "fibonacci test #3")
   (check-equal? (stream-ith fibonacci 3) 2 "fibonacci test #4")
   (check-equal? (stream-ith fibonacci 4) 3 "fibonacci test #5")
   (check-equal? (stream-ith fibonacci 5) 5 "fibonacci test #6")
   (check-equal? (stream-ith fibonacci 6) 8 "fibonacci test #7")
   (check-equal? (stream-ith fibonacci 7) 13 "fibonacci test #8")
   (check-equal? (stream-ith fibonacci 8) 21 "fibonacci test #9")

   ;; Interleave ;;
   (check-equal? (stream-ith (interleave fibonacci undecided) 0) 0 "interleave test #1
")
   (check-equal? (stream-ith (interleave fibonacci undecided) 3) #f "interleave test #
2")
   (check-equal? (stream-ith (interleave fibonacci undecided) 8) 3 "interleave test #3
")
   (check-equal? (stream-ith (interleave fibonacci undecided) 9) #t "interleave test #
4")
   (check-equal? (stream-ith (interleave test-nats test-nats) 0) 1 "interleave test #5
")
   (check-equal? (stream-ith (interleave test-nats test-nats) 1) 1 "interleave test #6
")
   (check-equal? (stream-ith (interleave test-nats test-nats) 2) 2 "interleave test #7
")
   (check-equal? (stream-ith (interleave test-nats test-nats) 3) 2 "interleave test #8
")
   (check-equal? (stream-ith (interleave test-nats test-nats) 4) 3 "interleave test #9
")

   ;; Not These Guys Again ;;

   ;; NOTE: These tests do not verify that your solution ;;
   ;; uses the suggested approach to implementation. ;;
   ;; You should do that yourself! ;;

   (check-equal? (stream-factorial 4) 24 "stream-factorial test #1")
   (check-equal? (stream-factorial 0) 1 "stream-factorial test #2")
   (check-equal? (stream-factorial 1) 1 "stream-factorial test #3")
   (check-equal? (stream-factorial 5) 120 "stream-factorial test #4")
   (check-equal? (stream-factorial 7) 5040 "stream-factorial test #5")

   ;; More Bananas ;;
   (check-equal? (stream-ith (repeats "banana") 0) "banana" "repeats test #1")
   (check-equal? (stream-ith (repeats "banana") 1) "bananabanana" "repeats test #2")
   (check-equal? (stream-ith (repeats "banana") 2) "bananabananabanana" "repeats test
#3")
   (check-equal? (stream-ith (repeats "aa") 4) "aaaaaaaaaa" "repeats test #4")

   ;; Newton's Back For More
   (check-= (stream-ith (sqrt-stream 2) 100) (sqrt 2) 0.0001 "sqrt-stream test #1")
   (check-= (stream-ith (sqrt-stream 100) 100) (sqrt 100) 0.0001 "sqrt-stream test #2"
)
   (check-= (stream-ith (sqrt-stream 5) 100) (sqrt 5) 0.0001 "sqrt-stream test #3")
   (check-= (stream-ith (sqrt-stream 9) 100) (sqrt 9) 0.0001 "sqrt-stream test #4")
   (check-= (stream-ith (sqrt-stream 25) 100) (sqrt 25) 0.0001 "sqrt-stream test #5")
   ;;; Macros ;;;

   ;; Perl Style ;;

   ;; NOTE: These tests do not verify that your solution has ;;
   ;; proper evaluation semantics with respect to side effects. ;;
   ;; You should do that yourself! ;;

   (check-equal? (perform #t unless #f) #t "perform test #1")
   (check-equal? (perform #t if #f) #f "perform test #2")
   (check-equal? (perform (+ 1 2) if null) 3 "perform test #3")
   (check-equal? (perform #f unless (+ 1 2)) 3 "perform test #4")
   (check-equal? (perform (+ 3 5) if (+ 1 2)) 8 "perform test #5")
   ))

(run-tests tests)
```

1 · flag

New post

To ensure a positive and productive discussion, please read our forum posting policies before posting.

B *I* Link <code> Pic Math Edit: Rich Preview

☐ Make this post anonymous to other students

☑ Subscribe to this thread at the same time

Add post