CS5010 - Problem Set 06 - Test Results

pdp-pair-kwong-wangzet

October 27, 2014

This test suite tests your implementation of Problem Set 05

1 File: outlines.rkt

Tests your implementation of outlines

1.1 Test-Group: Problem statement example (2 Points)

2/2

1.1.1 Test (equality)

This is the example that was given in the problem statement.
Input:

```
(nested-to-flat
'(("The first section"
   ("A subsection with no subsections")
   ("Another subsection"
    ("This is a subsection of 1.2")
    ("This is another subsection of 1.2"))
("The last subsection of 1"))
("Another section" ("More stuff") ("Still more stuff"))))
```

Expected Output:

```
'(((1) "The first section")
((1 1) "A subsection with no subsections")
((1 2) "Another subsection")
((1 2 1) "This is a subsection of 1.2")
((1 2 2) "This is another subsection of 1.2")
((1 3) "The last subsection of 1")
((2) "Another section")
((2 1) "More stuff")
((2 2) "Still more stuff"))
```

Expected Output Value:

```
(((1) "The first section")
((1 1) "A subsection with no subsections")
((1 2) "Another subsection")
((1 2 1) "This is a subsection of 1.2")
((1 2 2) "This is another subsection of 1.2")
((1 3) "The last subsection of 1")
((2) "Another section")
((2 1) "More stuff")
((2 2) "Still more stuff"))
```

Correct

1.2 Test-Group: Some more examples (3 Points)

3/3

1.2.1 Test (equality)

Input:

```
(nested-to-flat '(("Only one section here")))
```

Expected Output:

```
'(((1) "Only one section here"))
```

Expected Output Value:

```
(((1) "Only one section here"))
```

Correct

1.2.2 Test (equality, 1 partial points)

Input:

```
(nested-to-flat '(("First Section") ("Second Section")))
```

Expected Output:

```
'(((1) "First Section") ((2) "Second Section"))
```

Expected Output Value:

```
(((1) "First Section") ((2) "Second Section"))
```

Correct

1.2.3 Test (equality, 1 partial points)

Input:

```
(nested-to-flat
'(("One"
   ("One Point One"
    ("One Point One Point One"
     ("One Point One Point One Point One"
      ("One Point One Point One Point One Point One")))))))
```

Expected Output:

```
'(((1) "One")
((1 1) "One Point One")
((1 1 1) "One Point One Point One")
((1 1 1 1) "One Point One Point One Point One")
((1 1 1 1 1) "One Point One Point One Point One Point One"))
```

Expected Output Value:

```
(((1) "One")
((1 1) "One Point One")
((1 1 1) "One Point One Point One")
((1 1 1 1) "One Point One Point One Point One")
((1 1 1 1 1) "One Point One Point One Point One Point One"))
```

Correct

1.2.4 Test (equality, 1 partial points)

Input:

```
(nested-to-flat
'(("One"
   ("One Point One"
    ("One Point One Point One"
     ("One Point One Point One Point One"
      ("One Point One Point One Point One Point One")))))
("Two")))
```

Expected Output:

```
'(((1) "One")
((1 1) "One Point One")
((1 1 1) "One Point One Point One")
((1 1 1 1) "One Point One Point One Point One")
((1 1 1 1 1) "One Point One Point One Point One Point One")
((2) "Two"))
```

Expected Output Value:

```
(((1) "One")
((1 1) "One Point One")
((1 1 1) "One Point One Point One")
((1 1 1 1) "One Point One Point One Point One")
((1 1 1 1 1) "One Point One Point One Point One Point One")
((2) "Two"))
```

Correct

1.3 Test-Group: Test for nested-rep? (2 Points)

0.75/2

1.3.1 Test (equality, 0.25 partial points)

Simple Nested rep
Input:

```
(nested-rep? '(("One")))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

1.3.2 Test (equality)

Nat is an invalid nested rep
Input:

```
(nested-rep? 1)
```

Expected Output:

```
#f
```

Expected Output Value:

```
#f
```

Correct

1.3.3 Test (equality)

String is an invalid nested rep
Input:

```
(nested-rep? "One")
```

Expected Output:

```
#f
```

Expected Output Value:

```
#f
```

Correct

1.3.4 Test (equality, 0.5 partial points)

Problem set example
Input:

```
(nested-rep?
'(("The first section"
   ("A subsection with no subsections")
   ("Another subsection"
    ("This is a subsection of 1.2")
    ("This is another subsection of 1.2"))
("The last subsection of 1"))
("Another section" ("More stuff") ("Still more stuff"))))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

1.3.5 Test (equality, 0.25 partial points)

Flat rep is not a nested rep
Input:

```
(nested-rep?
'(("The first section"
   ("A subsection")
   ()
   ("A subsection after an empty one"))))
```

Expected Output:

```
#f
```

Expected Output Value:

```
#f
```

Wrong Output:

```
#t
```

1.3.6 Test (equality, 0.25 partial points)

Raw first section.
Input:

```
(nested-rep? '("First Section" ("A subsection")))
```

Expected Output:

```
#f
```

Expected Output Value:

```
#f
```

Wrong Output:

```
#t
```

1.3.7 Test (equality, 0.25 partial points)

No first section in the nested rep
Input:

```
(nested-rep? '((("No first section"))))
```

Expected Output:

```
#f
```

Expected Output Value:

```
#f
```

Wrong Output:

```
#t
```

1.3.8 Test (equality)

Empty is a valid nested outline
Input:

```
(nested-rep? empty)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

1.3.9 Test (equality)

No string in first section.
Input:

```
(nested-rep? '(()))
```

Expected Output:

```
#f
```

Expected Output Value:

```
#f
```

Wrong Output:

```
#t
```

2 File: pretty.rkt

Tests your implementation of pretty print for expr
Common Definitions

```
(define check-not-enough-room (lambda (x) (exn:fail? x)))


(define get-width
(lambda (l) (foldr max 0 (map (lambda (i) (string-length i)) l))))
```

2.1 Test-Group: Simple special cases (2 Points)

2/2

Common Definitions

```
(define SIMPLE-SUM (make-sum-exp (list 1337 42)))

(define SSUM-ONE-LINE-LIST (list "(+ 1337 42)"))

(define SSUM-ONE-LINE-WIDTH (string-length "(+ 1337 42)"))

(define SSUM-TWO-LINE-WIDTH (string-length "(+ 1337"))

(define SSUM-TWO-LINE-LIST (list "(+ 1337" "   42)"))

(define SIMPLE-MULT (make-mult-exp (list 74656 1701)))

(define SMULT-ONE-LINE-LIST (list "(* 74656 1701)"))

(define SMULT-TWO-LINE-LIST (list "(* 74656" "   1701)"))

(define SMULT-ONE-LINE-WIDTH (string-length "(* 74656 1701)"))

(define SMULT-TWO-LINE-WIDTH (string-length "(* 74656"))
```

2.1.1 Test (equality, 0.5 partial points)

Test for a single number rendering
Input:

```
(expr-to-strings 5 1)
```

Expected Output:

```
(list "5")
```

Expected Output Value:

```
("5")
```

Correct

2.1.2 Test (and, 0.5 partial points)

Simple sum
Test (equality)
Simple sum exp should come in a single list
Input:

```
(expr-to-strings SIMPLE-SUM SSUM-ONE-LINE-WIDTH)
```

Expected Output:

```
SSUM-ONE-LINE-LIST
```

Expected Output Value:

```
("(+ 1337 42)")
```

Correct
Test (error)
Simple sum exp cannot fit in given width
Input:

```
(expr-to-strings SIMPLE-SUM (- SSUM-TWO-LINE-WIDTH 1))
```

Expected Error should match:

```
check-not-enough-room
```

Correct

2.1.3 Test (and, 0.5 partial points)

Simple mult
Test (equality)
Simple mult exp in one line
Input:

```
(expr-to-strings SIMPLE-MULT SMULT-ONE-LINE-WIDTH)
```

Expected Output:

```
SMULT-ONE-LINE-LIST
```

Expected Output Value:

```
("(* 74656 1701)")
```

Correct
Test (error)
Simple mult exp does not fit in given width
Input:

```
(expr-to-strings SIMPLE-MULT (- SMULT-TWO-LINE-WIDTH 1))
```

Expected Error should match:

```
check-not-enough-room
```

Correct

2.2 Test-Group: Nested expressions (2 Points)

2/2

Common Definitions

```
(define EXPR
(make-mult-exp
 (list
  (make-sum-exp (list 1000 2000 3000))
  (make-sum-exp (list 50 60)))))

(define EXPR-ONE-LINE-LIST (list "(* (+ 1000 2000 3000) (+ 50 60))"))

(define EXPR-TWO-LINE-LIST
(list "(* (+ 1000 2000 3000)" "   (+ 50 60))"))

(define EXPR-FOUR-LINE-LIST
(list "(* (+ 1000" "      2000" "      3000)" "   (+ 50 60))"))

(define EXPR-FIVE-LINE-LIST
(list
 "(* (+ 1000"
 "      2000"
 "      3000)"
 "   (+ 50"
 "      60))"))
```

2.2.1 Test (equality, 0.25 partial points)

When given enough space, EXPR should be rendered on one line
Input:

```
(expr-to-strings EXPR (get-width EXPR-ONE-LINE-LIST))
```

Expected Output:

```
EXPR-ONE-LINE-LIST
```

Expected Output Value:

```
("(* (+ 1000 2000 3000) (+ 50 60))")
```

Correct

2.2.2 Test (equality, 0.5 partial points)

When given slightly not enough space to render EXPR on one line, it should be rendered in two lines
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-ONE-LINE-LIST) 1))
```

Expected Output:

```
EXPR-TWO-LINE-LIST
```

Expected Output Value:

```
("(* (+ 1000 2000 3000)" "   (+ 50 60))")
```

Correct

2.2.3 Test (equality, 0.5 partial points)

When rendered with the minimal possible width, the image of EXPR should have 5 lines
Input:

```
(expr-to-strings EXPR (get-width EXPR-FIVE-LINE-LIST))
```

Expected Output:

```
EXPR-FIVE-LINE-LIST
```

Expected Output Value:

```
("(* (+ 1000" "      2000" "      3000)" "   (+ 50" "      60))")
```

Correct

2.2.4 Test (error, 0.25 partial points)

When called with less than the minimal possible width, expr-to-strings should throw an error
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-FIVE-LINE-LIST) 1))
```

Expected Error should match:

```
check-not-enough-room
```

Correct

2.3 Test-Group: Complex expressions (3 Points)

2.5/3

Common Definitions

```
(define EXPR
(make-sum-exp
 (list
  (make-mult-exp (list 1))
  63450680
  (make-sum-exp (list 5 3))
  4
  (make-mult-exp (list 1234567890 67450))
  (make-sum-exp
   (list
    40
    (make-sum-exp
     (list
      (make-mult-exp (list 45830 5834))
      (make-mult-exp (list 56 6543))))))
1337)))


(define EXPR-ONE-LINE-LIST
(list
 "(+ (* 1) 63450680 (+ 5 3) 4 (* 1234567890 67450) (+ 40 (+ (* 45830
5834) (* 56 6543))) 1337)"))


(define EXPR-MAX-LINE-LIST
(list
 "(+ (* 1)"
 "   63450680"
 "   (+ 5 3)"
 "   4"
 "   (* 1234567890"
 "      67450)"
 "   (+ 40"
 "      (+ (* 45830"
 "            5834)"
 "         (* 56"
 "            6543)))"
 "   1337)"))


(define EXPR-SEVEN-LINE-LIST
(list
 "(+ (* 1)"
 "   63450680"
```

```
 "   (+ 5 3)"
 "   4"
 "   (* 1234567890 67450)"
 "   (+ 40 (+ (* 45830 5834) (* 56 6543)))"
 "   1337)"))


(define EXPR-EIGHT-LINE-LIST
(list
 "(+ (* 1)"
 "   63450680"
 "   (+ 5 3)"
 "   4"
 "   (* 1234567890 67450)"
 "   (+ 40"
 "      (+ (* 45830 5834) (* 56 6543)))"
 "   1337)"))


(define EXPR-NINE-LINE-LIST
(list
 "(+ (* 1)"
 "   63450680"
 "   (+ 5 3)"
 "   4"
 "   (* 1234567890 67450)"
 "   (+ 40"
 "      (+ (* 45830 5834)"
 "         (* 56 6543)))"
 "   1337)"))


(define EXPR-ELEVEN-LINE-LIST
(list
 "(+ (* 1)"
 "   63450680"
 "   (+ 5 3)"
 "   4"
 "   (* 1234567890"
 "      67450)"
 "   (+ 40"
 "      (+ (* 45830"
 "            5834)"
 "         (* 56 6543)))"
 "   1337)"))
```

2.3.1 Test (equality, 0.25 partial points)

The width of EXPR on one line should be equal to the width of the image of it's string representation on one line
Input:

```
(expr-to-strings EXPR (get-width EXPR-ONE-LINE-LIST))
```

Expected Output:

```
EXPR-ONE-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1) 63450680 (+ 5 3) 4 (* 1234567890 67450) (+ 40 (+ (* 45830
5834) (* 56 6543))) 1337)")
```

Correct

2.3.2 Test (equality, 0.25 partial points)

Rendering the image of EXPR with a limit slightly smaller than the maximum width should yield seven lines
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-ONE-LINE-LIST) 1))
```

Expected Output:

```
EXPR-SEVEN-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1)"
"   63450680"
"   (+ 5 3)"
"   4"
"   (* 1234567890 67450)"
"   (+ 40 (+ (* 45830 5834) (* 56 6543)))"
"   1337)")
```

Correct

2.3.3 Test (equality, 0.5 partial points)

An image of EXPR created with bounds that are only slightly too narrow to render it on 7 lines should have 8 lines.
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-SEVEN-LINE-LIST) 1))
```

Expected Output:

```
EXPR-EIGHT-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1)"
"   63450680"
"   (+ 5 3)"
"   4"
"   (* 1234567890 67450)"
"   (+ 40"
"      (+ (* 45830 5834) (* 56 6543)))"
"   1337)")
```

Correct

2.3.4 Test (equality, 0.5 partial points)

An image of EXPR created with bounds that are only slightly too narrow to render it on 8 lines should have 9 lines.
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-EIGHT-LINE-LIST) 1))
```

Expected Output:

```
EXPR-NINE-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1)"
"   63450680"
"   (+ 5 3)"
"   4"
"   (* 1234567890 67450)"
"   (+ 40"
"      (+ (* 45830 5834)"
"         (* 56 6543)))"
"   1337)")
```

Correct

2.3.5 Test (equality, 0.5 partial points)

An image of EXPR created with bounds that are only slightly too narrow to render it on 9 lines should have 11 lines.
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-NINE-LINE-LIST) 1))
```

Expected Output:

```
EXPR-ELEVEN-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1)"
"   63450680"
"   (+ 5 3)"
"   4"
"   (* 1234567890"
"      67450)"
"   (+ 40"
"      (+ (* 45830"
"            5834)"
"         (* 56 6543)))"
"   1337)")
```

Correct

2.3.6 Test (equality, 0.25 partial points)

Rendering EXPR with the minimal possible width should result in an image with 12 lines
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-ELEVEN-LINE-LIST) 1))
```

Expected Output:

```
EXPR-MAX-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1)"
"   63450680"
"   (+ 5 3)"
"   4"
"   (* 1234567890"
"      67450)"
"   (+ 40"
```

```
"      (+ (* 45830"
"            5834)"
"         (* 56"
"            6543)))"
"   1337)")
```

Correct

2.3.7 Test (error, 0.25 partial points)

Trying to render EXPR with less than the minimal possible width should result in an error
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-MAX-LINE-LIST) 1))
```

Expected Error should match:

```
check-not-enough-room
```

Correct

2.3.8 Test (equality, 0.25 partial points)

Render an expr with just enough room for a parenthese.
Input:

```
(expr-to-strings
(make-sum-exp
 (list 1234 567 (make-sum-exp (list (make-sum-exp (list 1 1)) 1))))
13)
```

Expected Output:

```
(list "(+ 1234" "   567" "   (+ (+ 1 1)" "      1)))")
```

Expected Output Value:

```
("(+ 1234" "   567" "   (+ (+ 1 1)" "      1)))")
```

Wrong Output:

```
("(+ 1234" "   567" "   (+ (+ 1 1)" "      1))")
```

2.4 Test-Group: The Two Column Bug (1 Points)

1/1

Common Definitions

```
(define EXPR
(make-sum-exp
 (list
  (make-mult-exp (build-list 20 add1))
  (make-mult-exp (build-list 20 add1)))))


(define EXPR-ONE-LINE-LIST
(list
 "(+ (* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20) (* 1
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20))"))


(define EXPR-TWO-LINE-LIST
(list
 "(+ (* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)"
 "   (* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20))"))


(define EXPR-MAX-LINE-LIST
(list
 "(+ (* 1"
 "      2"
 "      3"
 "      4"
 "      5"
 "      6"
 "      7"
 "      8"
 "      9"
 "      10"
 "      11"
 "      12"
 "      13"
 "      14"
 "      15"
 "      16"
 "      17"
 "      18"
 "      19"
 "      20)"
 "   (* 1"
 "      2"
 "      3"
```

```
 "      4"
 "      5"
 "      6"
 "      7"
 "      8"
 "      9"
 "      10"
 "      11"
 "      12"
 "      13"
 "      14"
 "      15"
 "      16"
 "      17"
 "      18"
 "      19"
 "      20))"))

(define EXPR-21-LINE-LIST
(list
 "(+ (* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)"
 "   (* 1"
 "      2"
 "      3"
 "      4"
 "      5"
 "      6"
 "      7"
 "      8"
 "      9"
 "      10"
 "      11"
 "      12"
 "      13"
 "      14"
 "      15"
 "      16"
 "      17"
 "      18"
 "      19"
 "      20))"))
```

2.4.1 Test (equality)

Enough room to render on one line
Input:

```
(expr-to-strings EXPR (get-width EXPR-ONE-LINE-LIST))
```

Expected Output:

```
EXPR-ONE-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20) (* 1
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20))")
```

Correct

2.4.2 Test (equality)

An image of EXPR created with bounds that are only slightly too narrow to render it on 1 line should have 2 lines.
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-ONE-LINE-LIST) 1))
```

Expected Output:

```
EXPR-TWO-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)"
"   (* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20))")
```

Correct

2.4.3 Test (equality)

An image of EXPR created with bounds that are only slightly too narrow to render it on 2 lines should have 21 lines.
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-TWO-LINE-LIST) 1))
```

Expected Output:

```
EXPR-21-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)"
"   (* 1"
"      2"
"      3"
"      4"
"      5"
"      6"
"      7"
"      8"
"      9"
"      10"
"      11"
"      12"
"      13"
"      14"
"      15"
"      16"
"      17"
"      18"
"      19"
"      20))")
```

Correct

2.4.4 Test (equality)

An image of EXPR created with bounds that are only slightly too narrow to render it on 21 lines should have 40 lines.
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-21-LINE-LIST) 1))
```

Expected Output:

```
EXPR-MAX-LINE-LIST
```

Expected Output Value:

```
("(+ (* 1"
"      2"
"      3"
"      4"
"      5"
"      6"
"      7"
"      8"
"      9"
```

```
"      10"
"      11"
"      12"
"      13"
"      14"
"      15"
"      16"
"      17"
"      18"
"      19"
"      20)"
"   (* 1"
"      2"
"      3"
"      4"
"      5"
"      6"
"      7"
"      8"
"      9"
"      10"
"      11"
"      12"
"      13"
"      14"
"      15"
"      16"
"      17"
"      18"
"      19"
"      20))")
```

Correct

2.4.5 Test (error)

Trying to render EXPR in less than minimal possible width will result in error
Input:

```
(expr-to-strings EXPR (- (get-width EXPR-MAX-LINE-LIST) 1))
```

Expected Error should match:

```
check-not-enough-room
```

Correct

3 Results

Successes: 31
Wrong Outputs: 5
Errors: 0
Achieved Points: 13.25
Total Points (rounded): 13.0/15