Metody Programowania lista 1

Szymon Kopa

20 lutego 2018

1 Ćwiczenie 1

1.1

```
>10
10
```

1.2

```
>(+ 5 3 4)
12
```

1.3

```
>(- 9 1)
8
```

1.4

```
>(/ 6 2)
3
```

1.5

```
>(+ (* 2 4) (- 4 6) )

6
```

1.6

```
>(define a 3)
>( define b (+ a 1) )
>(+ a b (* a b ) )

19
```

1.7

```
>(= a b )

#f
```

1.8

```
>( if ( and (> b a ) (< b (* a b ) ) ) b a)

4
```

1.9

```
>( cond [(= a 4) 6]
[(= b 4) (+ 6 7 a ) ]
[ else 25])

16
```

1.10

```
>(+ 2 ( if (> b a ) b a ) )

6
```

1.11

```
>(* ( cond [( > a b ) a ]
[(< a b ) b ]
[ else
-1])
(+ a 1) )

16
```

2 Ćwiczenie 2

$$\frac{5+4+(2-(3-(6+\frac{4}{5})))}{3(6-2)(2-7)}$$

```
    (/ ({+ 5 4 (- 2 {- 3 (+ 6 {/ 4 5})})}) {* 3 (- 6 2 ) (- 2 7 )})
```

3 Ćwiczenie 3

4 Ćwiczenie 4

```
#lang racket
(define (sum-of-squares a b)
  (+ (* a a) (* b b)))
(define (max a b)
  (cond [(> a b) a]
        [else b]))
(define (sum-of-squares-of-2-largest a b c)
  (cond [(> a b) (sum-of-squares a (max b c))]
        [else (sum-of-squares b (max a c))]))
```

5 Ćwiczenie 5

Mamy wyjaśnić:

```
( define ( a-plus-abs-b a b )
  (( if (> b 0) + -) a b ) )
```

Skoro wiemy, że naszymi operatorami mogą być wyrażenia złożone. Jeśli b będzie ujemne to od a będziemy odejmować b. Wyjdzie nam z tego wartość bezwzględna. Operatorem złożonym jest całe wyrażenie z "if". Z argumentami przykładowymi 2 i -2 mamy:

```
( a-plus-abs-b 2 -2 )
(( if (> -2 0) + -) 2 -2 )


((if #f + -)2 -2)
(2 -2)


4
```

6 Ćwiczenie 6

```
(and #f (< 0 1))

(or #t (< 0 1))
```

7 Ćwiczenie 7

```
(test 0 (p))
```

Jest to język gorliwy, ponieważ program się zapętla przy wywołaniu powyższej procedury. Chcemy cały czas obliczyć p, które wywołuje w pętli samą siebie.

8 Ćwiczenie 8

Chcemy znaleźć sufit z logarytmu.

```
(define (power-close-to b n)
  (define (pct e)
    (if (> (exp b e)n)
    e
    (pct (+ e 1) )
  ))
  (pct 0))
```

9 Zadanie domowe

Praktycznie kopiujemy z wykładu, sami możemy przyjąć jakąś dokładność. Zmieniamy tylko approxa na podanego w zadaniu oraz obliczanie kwadratu na obliczanie sześcianu. Tyle, darmowa pracownia. Dla bignumów oraz licz bliskim 0 się krzaczy ze względu na *utratę cyfr znaczących*.

```
#lang racket
(define (cube-root x)
  (define (dist x y)
    (abs (- x y)))

  (define (cube x)
    (* x x x))

  (define (improve approx)
    (/ (+ (/ x (* approx approx)) (* 2 approx)) 3))

  (define (good-enough? approx)
    (< (dist x (cube approx)) 0.00001))

  (define (iter approx)
    (cond
      [(good-enough? approx) approx]
      [else                (iter (improve approx))]))

  (iter 1.0))

(define (test x)
    (<= (abs (- (expt x (/ 1 3)) (cube-root x))) 0.00001))

(cube-root 27) (expt 27 (/ 1 3)) (test 27)
(cube-root 0) (expt 0 (/ 1 3)) (test 0)
(cube-root (expt 10 6)) (expt (expt 10 6) (/ 1 3)) (test (expt 10 6))
(cube-root 1) (expt 1 (/ 1 3)) (test 1)
(cube-root 99) (expt 99 (/ 1 3)) (test 99)
```

```
(cube-root 0) (expt 0 (/ 1 3)) (test 0)
(cube-root 98765) (expt 98765 (/ 1 3)) (test 98765)
(cube-root 999999) (expt 999999 (/ 1 3)) (test 999999)
(cube-root 2.5) (expt 2.5 (/ 1 3)) (test 2.5)
(cube-root 1.1) (expt 1.1 (/ 1 3)) (test 1.1)
(cube-root 111.111) (expt 111.111 (/ 1 3)) (test 111.111)
(cube-root 13) (expt 13 (/ 1 3)) (test 13)
(cube-root 0.1) (expt 0.1 (/ 1 3)) (test 0.1)
```