

Lista zagadnień nr 1

Ćwiczenie 1.

Przeanalizuj poniższą sekwencję wyrażeń. Jaki wynik wypisze interpreter w odpowiedzi na każde z nich, zakładając, że będą obliczane w kolejności w której są podane? Sprawdź swoje przewidywania używając interpretera.

```
10

(+ 5 3 4)

(- 9 1)

(/ 6 2)

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

(define a 3)

(define b (+ a 1))

(+ a b (* a b))

(= a b)

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

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

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

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

Ćwiczenie 2.

Przedstaw w postaci prefiksowej poniższe wyrażenie:

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

Ćwiczenie 3.

Zastosuj zasady obliczania wyrażeń poznane na wykładzie do obliczenia wartości poniższych wyrażeń. Które z nich spowodują błąd i dlaczego?

`(* (+ 2 2) 5)`

`(* (+ 2 2) (5))`

`(* (+ (2 2) 5))`

`(* (+ 2
2) 5)`

`(5 * 4)`

`(5 * (2 + 2))`

`((+ 2 3))`

`+`

`(define + (* 2 3))`

`+`

`(* 2 +)`

`(define (five) 5)`

`(define four 4)`

`(five)`

`four`

`five`

`(four)`

Ćwiczenie 4.

Zdefiniuj procedurę o trzech argumentach będących liczbami, której wynikiem jest suma kwadratów dwóch większych jej argumentów.

Ćwiczenie 5.

Racket wyposażony jest w wieloargumentowe formy specjalne `and` i `or`, które obliczają podwyrażenia od lewej do prawej tak długo, aż trafi odpowiednio na wartość `false` lub `true`, i nieobliczające pozostałych podwyrażeń. Podaj przykład wyrażenia, którego obliczanie zakończyłoby się błędem, *gdyby* `and` *był* *procedurą wbudowaną*, a nie formą specjalną. Znajdź analogiczny przykład dla formy `or`.

Ćwiczenie 6.

Zauważ że w naszym modelu obliczania wartości dopuszczamy, aby operatorami były wyrażenia złożone. Korzystając z tej obserwacji, wyjaśnij działanie następującej procedury:

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

Ćwiczenie 7.

Zdefiniuj procedurę `power-close-to`, która przyjmuje jako argumenty liczby dodatnie b i n , i zwraca najmniejszą liczbę całkowitą e taką, że $b^e > n$. Możesz użyć wbudowanej procedury `expt` podnoszącej liczbę do danej potęgi.

```
(power-close-to 2 1000)
> 10
```

```
(expt 2 10)
> 1024
```

Użyj struktury blokowej, aby ukryć definicje pomocniczych procedur przed użytkownikiem, i użyj lokalnego wiązania zmiennych aby usunąć zbędne parametry.

Zadanie domowe (na pracownię)

Metodę Newtona, omówioną na wykładzie dla przykładu pierwiastka kwadratowego, można zastosować również do obliczania pierwiastka sześciennego. W

tym celu wykorzystujemy fakt, że jeśli y jest przybliżoną wartością pierwiastka sześciennego z x , to

$$\frac{\frac{x}{y^2} + 2y}{3}$$

jest lepszym przybliżeniem. Korzystając z tej zależności zaimplementuj procedurę `cube-root`, analogiczną do procedury obliczającej pierwiastki kwadratowe. Pamiętaj, aby użyć struktury blokowej i wiązania składni, żeby uzyskać zwięzły i czytelny kod, a także aby ukryć przed użytkownikiem pomocnicze procedury!

Przetestuj też działanie swojej procedury na kilku przykładach. Możesz to zrobić np. zamieszczając kilka przykładowych wyrażeń na końcu pliku. Czy Twoja procedura działa poprawnie dla *wszystkich* poprawnych danych wejściowych?¹

Uwaga! Plik o nazwie `solution.rkt` zawierający definicję procedury `cube-root` i przykłady testowe należy przesłać w systemie Web-CAT dostępnym na SKOS-owej stronie przedmiotu w *nieprzekraczalnym* terminie **8 marca 2021 r., godz. 05.30**. W pliku zamieść dodatkową klauzulę

```
(provide cube-root)
```

Sprawia ona, że procedura jest widoczna dla sprawdzaczki (dostępny też jest szablon rozwiązania na SKOS-ie). Pamiętaj o zasadach współpracy opisanych w regulaminie przedmiotu.

¹W tym tygodniu do zaliczenia zadania nie jest wymagane *gruntowne* przetestowanie definiowanej procedury, ale *jest wymagana* poprawna i czytelna struktura kodu.