Programowanie funkcyjne — kolokwium nr 1, 10.12.2014

Instrukcja: Rozwiązania zadań należy przesłać do godziny 9:50 na adres `kolokwium.pf@gmail.com` (decyduje data stempla googlowego). Każde zadanie należy przesłać w oddzielnym pliku Zadanie$n$.hs ($n = 1, 2, 3$). W rozwiązaniach nie można korzystać z modułów innych niż Prelude, tzn. niedozwolone jest użycie polecenia import. Punktacja: po 10 punktów za zadanie.

Zadanie 1. Napisać bezpunktowo funkcję numocc, która zlicza wystąpienia wskazanego elementu w podanych listach, tzn. numocc $x\ [\ell_1, \ell_2, ...\ell_n] = [a_1, a_2, ..., a_n]$, gdzie a_i to liczba wystąpień x w liście ℓ_i. Na przyklad

numocc $1\ [[1, 2], [2, 3, 2, 1, 1], [3]] = [1, 2, 0]$.

Podać najogólniejszą możliwą sygnaturę.

Zadanie 2. Obliczenia z użyciem operacji dodawania, mnożenia i negacji na elementach typu a (będącego instancją klasy Num) będziemy reprezentować przez drzewo obliczeń typu CT a, zdefiniowane jako

data CT a = Empty | Leaf a | Join (CT a) Op (CT a),

przy czym Empty to drzewo puste, Leaf — liść drzewa zawierający wartość typu a, zaś Join — drzewo reprezentujące wykonanie operacji Op na obliczeniach z lewego i prawego poddrzewa. Typ Op jest określony jako

data Op = Add | Mul | Neg.

(a) Napisać funkcję wf :: CT $a \to$ Bool, która dla podanego drzewa sprawdza, czy jest ono poprawne, w tym sensie, że nie jest puste i nie zawiera poddrzew postaci (Join Empty Add r) lub (Join ℓ Add Empty) i analogicznie dla Mul, oraz poddrzew postaci (Join ℓ Neg r), gdzie oba drzewa ℓ, r są niepuste.

(b) Napisać funkcję eval :: Num $a \Rightarrow$ CT $a \to a$, która dla podanego drzewa wyliczy wartość reprezentowanego w nim obliczenia. Na przykład dla

T = (Join (Join (Leaf 3) Add (Leaf 2)) Mul (Join (Leaf 2) Neg Empty))

wartość eval T = -10. Jeśli drzewo nie jest poprawne, funkcja powinna sygnalizować błąd.

Zadanie 3. Napisać bezpunktowo funkcję h, która zwraca elementy podanej listy znajdujące się na pozycjach o numerach parzystych, w takiej kolejności, w jakiej występują w oryginalnej liście. Na przykład

h $[0, 1, 2, 3, 4] = [0, 2, 4]$,
h "AlaMaKota" = "Aaaoa".

Programowanie funkcyjne — kolokwium nr 1, 10.12.2015

Instrukcja: Rozwiązania zadań należy przesłać do godziny 13:15 na adres `kolokwium.pf@gmail.com` (decyduje data stempla googlowego). Każde zadanie należy przesłać w oddzielnym pliku: Zadanie1.hs, Zadanie2.hs i Zadanie3.hs (jeśli zadania nie udało się rozwiązać, należy przesłać pusty plik). Plików nie należy zipować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe (niedozwolone jest użycie polecenia `import`). Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie.
Uwaga: Korzystanie z internetu poza wyznaczonym czasem skutkuje automatycznym otrzymaniem 0 punktów.

Zadanie 1. Każde słowo złożone z liter a i b przekształcamy idąc od lewej według następujących reguł: $ab \to a, ba \to b, bb \to a, aa \to aaa$ (jeśli słowo ma nieparzystą długość, ostatnią literę przepisujemy). Napisać funkcję `dlugosc`, zwracającą liczbę iteracji powyższych reguł, które prowadzą do słowa złożonego z samych liter a lub słowa długości mniejszej niż 2. Przykładowo:

`dlugosc "abaaa" = 1`, bo $|ab|aa|a \to |a|aaa|a$
`dlugosc "abba" = 2`, bo $|ab|ba| \to |a|b|$ oraz $|ab| \to a$
`dlugosc "babba" = 3`, bo $|ba|bb|a \to |b|a|a$, $|ba|a \to |b|a$ oraz $|ba| \to b$

Zadanie 2. (a) Dla dwóch liczb całkowitych $a, b > 1$ oznaczamy przez $\mathrm{val}(a, b)$ największą potęgę liczby b, która dzieli a, np. $\mathrm{val}(56, 2) = 3$, bo 2^3 dzieli 56, ale 2^4 nie dzieli 56; podobnie $\mathrm{val}(56, 3) = 0$, bo 3^1 nie dzieli 56. Napisać funkcję:

```
val :: Integer -> Integer -> Integer
```

która dla podanych a i b oblicza $\mathrm{val}(a, b)$.
(b) Za pomocą powyższej funkcji `val` napisać funkcję:

```
g :: Integer -> Integer -> [Integer]
```

która dla podanych $k > 1$ oraz $v \geqslant 0$ zwraca listę (nieskończoną, w dowolnej kolejności) liczb naturalnych $n > 1$, takich że $\mathrm{val}(n, k) = v$. Przykładowo:

```
g 2 0 = [3,5,7,9,...]
g 3 1 = [3,6,12,15,21,...]
```

Zadanie 3. Kłosem nazywamy strukturę danych o funkcjonalności przypominającej listę, która umożliwia dokładanie elementów na początek i na koniec w czasie stałym, a ponadto odczytanie elementów po kolei. Stworzyć typ `Klos a` przechowujący elementy typu `a`. Zdefiniować funkcje:

```
wnpk :: Klos a -> a -> Klos a
wnkk :: Klos a -> a -> Klos a
k2list :: Klos a -> [a]
```

Funkcje mają, odpowiednio, wstawiać element na początek i na koniec kłosa oraz zamieniać kłos na listę. W ostatniej funkcji nie nakładamy ograniczenia na złożoność.

Programowanie funkcyjne — kolokwium nr 1, 7.12.2016

Instrukcja: Rozwiązania zadań należy przesłać do godziny 9:30 na adres `kolokwium.pf@gmail.com` (decyduje data stempla googlowego). Każde zadanie należy przesłać w oddzielnym pliku: zadanie1.hs, zadanie2.hs i zadanie3.hs. Plików nie należy zipować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe; niedozwolone jest użycie polecenia `import`. Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie. Uwaga: korzystanie z internetu poza wyznaczonym czasem skutkuje automatycznym otrzymaniem 0 punktów.

Zadanie 1. Napisać funkcję pownum :: Integer → [Integer], która dla podanego $n \geqslant 1$ zwraca (być może nieskończoną) listę składającą się ze wszystkich liczb takich, że suma n-tych potęg cyfr równa jest tej liczbie. Zatem np.

```
pownum 5 = [1, 4150, ...]
```

gdyż $4^5 + 1^5 + 5^5 + 0^5 = 4150$.

Zadanie 2. Napisać funkcję ps, która dla podanej listy zwraca listę zawierającą wszystkie jej prefiksy — w kolejności od najkrótszego (jednoelementowego) do najdłuższego (cała lista) — a następnie kolejno coraz krótsze sufiksy tej listy. Przykładowo:

```
ps "Test" = ["T", "Te", "Tes", "Test", "est", "st", "t"]
ps [3,5,2] = [[3], [3,5], [3,5,2], [5,2], [2]]
```

Dla pustej listy wynik może być dowolny. W rozwiązaniu należy w istotny sposób użyć funkcji foldl lub foldr.

Zadanie 3. Rododendronem nazywamy drzewo, w którym każdy wierzchołek może mieć dowolną liczbę potomków (być może zero). Rododendron nie może być pusty. Stworzyć typ Rd a, przechowujący elementy typu a w rododendronie, i zdefiniować funkcje:

```
el :: Eq a => Rd a -> a -> Bool
subst :: Eq a => a -> a -> Rd a -> Rd a
rd2list :: Rd a -> [a]
```

Funkcje mają, odpowiednio: sprawdzać czy podany element należy do rododendronu, zamieniać wszystkie wystąpienia pierwszego podanego elementu na drugi oraz zamieniać rododendron na listę zgodnie z porządkiem preorder.

Programowanie funkcyjne — kolokwium nr 1, 6.12.2017

Instrukcja: Rozwiązania zadań należy przesłać do godziny 11:00 na adres `kolokwium.pf@gmail.com` (decyduje data stempla googlowego). Każde zadanie należy przesłać w oddzielnym pliku: zadanie1.hs, zadanie2.hs i zadanie3.hs. Plików nie należy zipować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe; niedozwolone jest użycie polecenia `import`. Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie. Uwaga: korzystanie z internetu poza wyznaczonym czasem skutkuje automatycznym otrzymaniem 0 punktów.

Zadanie 1. Napisać funkcję rd :: Integer → [Integer], która dla podanego $n \geqslant 1$ zwraca (być może nieskończoną) listę składającą się ze wszystkich liczb k takich, że suma dodatnich dzielników liczby k równa jest $n + k$. Zatem np. rd 1 zwraca listę wszystkich liczb pierwszych, rd 2 daje listę pustą, a rd 3 to lista zawierająca tylko liczbę 4.

W przypadku, gdy wynik jest listą skończoną, wskazane jest, by funkcja kończyła działanie. Wskazówka: jeśli liczba k ma dzielnik d, to ma również dzielnik k/d. Na tej podstawie można oszacować, że dla $n > 1$ w wyniku funkcji rd n nie może być liczb większych niż n^2.

Zadanie 2. Napisać funkcję repl :: Eq $a \Rightarrow [a] \to [(a, a)] \to [a]$, która dla danej listy ℓ oraz listy par dokonuje zamian elementów w ℓ w taki sposób, że każde wystąpienie pierwszego elementu pewnej pary zostaje zamienione na drugi element tej pary, np.

```
repl [1,2,3,1,2] [(2,4)] = [1,4,3,1,4]
repl "alamakota" [('a','u'), ('o','e')] = "ulumuketu"
```

W rozwiązaniu należy w istotny sposób użyć funkcji foldl lub foldr. Uwaga: można założyć, że wszystkie elementy występujące w parach są różne, tzn. lista k par $(a_1, b_1), ..., (a_k, b_k)$ zawiera $2k$ różnych elementów $a_1, ..., a_k, b_1, ..., b_k$.

Zadanie 3. Słabym drzewem binarnym nazywamy strukturę, w której każdy wierzchołek zawiera pewną wartość, a ponadto ma 0, 1 lub 2 wierzchołki potomne. W tym ostatnim przypadku kolejność potomków nie jest rozróżniana, tzn. drzewa różniące się jedynie kolejnością potomków uznajemy za równe. Słabe drzewo binarne nie może być puste. Stworzyć typ Sdb a, przechowujący elementy typu a w słabym drzewie binarnym, i zdefiniować funkcje:

```
el :: Eq a => Sdb a -> a -> Bool
eq :: Eq a => Sdb a -> Sdb a -> Bool
sdb2list :: Sdb a -> [a]
```

Funkcje mają, odpowiednio: sprawdzać czy podany element należy do drzewa, sprawdzać czy podane drzewa są równe oraz zamieniać drzewo na listę przeszukując je wszerz (kolejność przeglądania potomków może być dowolna).

Programowanie funkcyjne — kolokwium nr 1, 4.12.2018

Instrukcja: Rozwiązania należy przesłać do godziny 9:45, w jednym mailu, na adres kolokwium.pf@gmail.com. Każde zadanie należy przesłać w oddzielnym pliku: zadanie1.hs, zadanie2.hs i zadanie3.hs. Plików nie należy zipować. Rozwiązania muszą się poprawnie kompilować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe; niedozwolone jest użycie polecenia import. Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie. Uwaga: korzystanie z internetu poza wyznaczonym czasem skutkuje automatycznym otrzymaniem 0 punktów.

Zadanie 1. Rozważmy ciągi $(a_n), (b_n)$, które spełniają następującą zależność rekurencyjną

$$\begin{aligned} a_n &= (n-1)b_{n-1} - 3a_{n-1} \\ b_n &= 3b_{n-1} + (n-1)^2 a_{n-1} - (n-1)^2 \end{aligned}$$

Dodatkowo wiemy, że $a_0 = b_0 = 1$. Napisać funkcję seqIndex m, która zwraca *najmniejsze* k takie, że $a_0 + a_1 + \ldots + a_k \geqslant m$. Na przykład seqIndex 100 = 4, seqIndex 1000000 = 8.

Zadanie 2. Rozważmy typ danych

```
data Expr a = Value a
            | Add (Expr a) (Expr a)
            | Mul (Expr a) (Expr a)
            | Sub (Expr a) (Expr a)
            | P
```

przechowujący *częściowe* wyrażenia, tzn. wyrażenia, które zawierają operacje dodawania (Add), mnożenia (Mul) i odejmowania (Sub) oraz wartość P, która oznacza, iż konkretny argument nie jest jeszcze znany. Argumenty do operacji arytmetycznych przechowujemy za pomocą Value. Napisać funkcję eq :: (Eq a) $\rightarrow$ Expr $a \rightarrow$ Expr $a \rightarrow$ Bool, która zwraca True, jeśli wyrażenia są takie same, i False w przeciwnym wypadku. Przyjmujemy, że dwa wyrażenia są takie same, jeśli jedno można otrzymać z drugiego przez zamiany P na dowolne inne wyrażenia. Na przykład

```
eq (Add (Value 1) (Value 2)) (Add (Value 1) (Value 3)) = False
eq (Add (Value 1) (Value 2)) P = True
```

Zadanie 3. Napisać funkcję cykl, która dla podanej niepustej listy zwraca listę jej wszystkich przesunięć cyklicznych, w dowolnej kolejności. Podać najogólniejszą możliwą sygnaturę. Funkcja ma w nietrywialny sposób korzystać z foldl lub foldr, przy czym fold musi stanowić najbardziej zewnętrzną część definicji, np. cykl ℓ = foldl ... Przykładowo, wywołanie cykl [1,2,3] powinno zwrócić [[1,2,3], [2,3,1], [3,1,2]] lub dowolną permutację takiej listy.

Programowanie funkcyjne — kolokwium nr 1, 27.11.2019

Instrukcja: Rozwiązania należy przesłać do godziny 9:45, w jednym mailu, na adres kolokwium.pf@gmail.com. Każde zadanie należy przesłać w oddzielnym pliku: zadanie1.hs, zadanie2.hs i zadanie3.hs. Plików nie należy zipować. Rozwiązania muszą się poprawnie kompilować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe; niedozwolone jest użycie polecenia import. Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie. Uwaga: korzystanie z internetu poza wyznaczonym czasem skutkuje automatycznym otrzymaniem 0 punktów.

Zadanie 1. Dla liczby naturalnej w każdym kroku tworzymy liczbę będącą sumą jej cyfr (w zapisie dziesiętnym) i postępujemy tak, aż dojdziemy do liczby jednocyfrowej. Napisać funkcję sevens :: Int → [Int], zwracającą n pierwszych liczb, dla których powyższy proces skończy się na liczbie 7. Na przykład sevens 4 = [7, 16, 25, 34]. Można też zauważyć, że do sevens 1000 należy np. liczba 8881, bo 8881 → 25 → 7.

Zadanie 2. Palindrom nazywamy *zbalansowanym*, jeżeli składa się on wyłącznie z liter a i b, i liczba liter a jest równa liczbie liter b. Napisać funkcję bp :: Int → [String], która dla danego n zwróci listę zbalansowanych palindromów o długości n.

Zadanie 3. *Miłorząb* to struktura danych o funkcjonalności przypominającej listę, która umożliwia: dokładanie elementów na początek i na koniec w czasie stałym, odczytanie elementów po kolei w czasie liniowym względem liczby elementów oraz usunięcie wszystkich elementów dołożonych na początek („urwanie lewego listka") i na koniec („urwanie prawego listka") w czasie stałym (nie liczymy czasu potrzebnego na zwolnienie pamięci). Stworzyć typ Mb a, przechowujący elementy typu a w miłorzębie, i zdefiniować funkcje:

```
dnp :: Mb a -> a -> Mb a
dnk :: Mb a -> a -> Mb a
mb2list :: Mb a -> [a]
ull :: Mb a -> Mb a
upl :: Mb a -> Mb a
```

Funkcje mają, odpowiednio, wstawiać element na początek i na koniec miłorzębu, zamieniać miłorząb na listę (z elementami w odpowiedniej kolejności, tzn. od początku miłorzębu do końca) oraz urywać lewy i prawy listek. Złożoność funkcji powinna być taka, jak opisano wcześniej.

Programowanie funkcyjne — kolokwium nr 1, 10.12.2020

Instrukcja: Każde zadanie należy przesłać na Pegaza w oddzielnym pliku: zadanie1.hs, zadanie2.hs i zadanie3.hs. Plików nie należy zipować. Rozwiązania muszą się poprawnie kompilować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe; niedozwolone jest użycie polecenia import. Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie.

Zadanie 1. Niech `type Point = (Double, Double)`. Napisać funkcję

`minDist :: [Point] -> (Point, Point, Double)`,

która zwróci parę punktów najbliższych sobie na liście podanej jako argument, wraz z odległością między nimi. W przypadku większej liczby możliwych rozwiązań należy wybrać dowolne; także kolejność elementów w parze wynikowej nie ma znaczenia. W rozwiązaniu można użyć faktu, iż `(Double, Double)` jest w klasie `Eq`. Przykładowo,

`minDist [(1,5),(1,1),(10,1),(-4,1)] = ((1.0,1.0),(1.0,5.0),4.0)`.

Zadanie 2. Rozważmy następującą definicję drzewa:

`data Tree a = Empty | Node a (Tree a) (Tree a)`.

Napisać funkcję

`findPath :: Eq a => a -> Tree a -> [a]`,

która zwraca ścieżkę (w postaci listy) od korzenia drzewa do elementu podanego jako pierwszy argument. Proszę założyć, że w drzewie nie ma powtarzających się elementów. Na przykład dla

```
t=Node 10 (Node 5 (Node 4 Empty Empty) (Node 6 Empty Empty)) (Node 20
Empty Empty)
```

mamy `findPath 6 t = [10,5,6]` oraz `findPath 7 t = []`.

Zadanie 3. W permutacji liczb od 1 do n pozycję k nazywamy *zamykającą*, jeżeli wśród pozycji od 1 do k znajdują się wszystkie liczby od 1 do k. Napisać funkcję

`cp :: [Integer] -> [Integer]`,

która dla permutacji podanej jako lista wyliczy wszystkie jej pozycje zamykające. Przykładowo, `cp [1,2,3] = [1,2,3]`, zaś `cp [3,2,1] = [3]`.

Programowanie funkcyjne — kolokwium nr 1, 8.12.2021

Instrukcja: Każde zadanie należy przesłać na Pegaza w oddzielnym pliku: zadanie1.hs, zadanie2.hs i zadanie3.hs. Plików nie należy zipować. Rozwiązania muszą się poprawnie kompilować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe; niedozwolone jest użycie polecenia import. Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie.

Zadanie 1. Rozpatrzmy następujący typ reprezentujący graf skierowany:

`type DirectedGraph = ([Int], Int -> Int -> Bool).`

Pierwszy element pary to zbiór wierzchołków V, drugi to funkcja f taka, że dla dowolnych $x, y \in V$ mamy, że $f\ x\ y = \text{True} \iff xy$ jest (skierowaną) krawędzią z x do y. Napisać funkcję

`atDistance :: DirectedGraph -> Int -> Int -> [Int],`

która dla wywołania `atDistance` $g\ d\ v$ zwraca listę (być może z powtórzeniami) wszystkich wierzchołków, które w grafie g są osiągalne z v drogą długości dokładnie d. Dla przypomnienia: w drodze wierzchołki mogą się powtarzać, zaś długość drogi to liczba jej krawędzi.

Zadanie 2. Napisać funkcję o sygnaturze

`wIluListach :: Int -> [[Int]] -> [Int],`

która dla liczby całkowitej dodatniej n oraz listy $[l_1, l_2, ..., l_k]$ list liczb naturalnych z przedziału $\{1, ..., n\}$ sprawdza, w jak wielu listach $l_1, l_2, ..., l_k$ występują kolejne liczby $1, ..., n$. Inaczej mówiąc, funkcja ma zwrócić listę, której pierwszym elementem jest liczba list, w których występuje liczba 1, drugim — liczba list, w których występuje 2 itd. Przykładowo:

`wIluListach 7 [[1,2,3],[3,4,5]]` ma zwrócić `[1,1,2,1,1,0,0]`, ponieważ liczba 1 występuje w jednej liście, podobnie jak 2, 4 i 5; liczba 3 występuje w dwóch listach, a 6 i 7 — w żadnej,

`wIluListach 7 [[1,2,3],[3,4,5],[5,6,1],[1,7,4],[3,7,6],[2,7,5],[2,4,6]]` ma zwrócić `[3,3,3,3,3,3,3]`, bo każda z liczb od 1 do 7 występuje w dokładnie trzech listach.

Zadanie 3. *Wierzba rosochata* to struktura danych, która pozwala na: dołączenie gałęzi (tj. listy elementów), usunięcie ostatnio dołączonej gałęzi, dołączenie elementu do ostatnio dołączonej gałęzi, usunięcie ostatnio dołączonego elementu (pod warunkiem, że od ostatniego dołączenia elementu nie były wykonywane operacje na gałęziach), podanie liczby gałęzi oraz zamianę całej struktury na listę w taki sposób, że ostatnio dołączona gałąź pojawia się w liście przed pozostałymi gałęziami. Zamiana na listę powinna być wykonalna w czasie liniowym względem łącznej liczby elementów, natomiast pozostałe operacje — w czasie stałym. Zdefiniować typ Wr a, służący do przechowywania elementów typu a w wierzbie rosochatej, oraz następujące funkcje, realizujące opisane wyżej operacje z odpowiednią złożonością:

```
dg :: Wr a -> [a] -> Wr a
ug :: Wr a -> Wr a
de :: Wr a -> a -> Wr a
ue :: Wr a -> Wr a
lg :: Wr a -> Integer
wr2l :: Wr a -> [a]
```

Programowanie funkcyjne — kolokwium nr 1, 19.12.2013

Instrukcja: Rozwiązania zadań należy przesłać do godziny 11:40 na adres `kolokwium.pf@gmail.com` (decyduje data stempla googlowego). Każde zadanie należy przesłać w oddzielnym pliku Zadanie$n$.hs ($n = 1, 2, 3$). W rozwiązaniach nie można korzystać z modułów innych niż Prelude, tzn. niedozwolone jest użycie polecenia import. Punktacja: po 10 punktów za zadanie.

Zadanie 1. Napisać funkcję oddbins n, generującą listę wszystkich ciągów binarnych o długości n, w których liczba jedynek jest nieparzysta. Ciągi reprezentujemy w postaci list, zatem np.

oddbins $3 = [[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 1]]$.

Kolejność ciągów w liście nie ma znaczenia. Zakładamy, że $n \geqslant 1$.

Zadanie 2. Napisać funkcję diffsums :: [[Int]] $\rightarrow$ [[Int]], która z wejściowej listy usuwa listy o powtarzającej się sumie. Na przykład

diffsums $[[1, 2], [3, 4, 5], [3], [], [7, 5]] = [[1, 2], [3, 4, 5], []]$ lub $[[7, 5], [3], []]$ itp.

Każda suma z wejściowej listy ma być reprezentowana w liście wynikowej przez dokładnie jedną listę. W rozwiązaniu należy użyć funkcji foldl lub foldr. Kolejność w liście wynikowej nie ma znaczenia, ale kolejność w blokach ma zostać zachowana.

Zadanie 3. Napisać bezpunktowo funkcję compref, podającą długość najdłuższego zgodnego odcinka początkowego dwóch list, oraz podać najogólniejszą możliwą sygnaturę. Na przykład

compref $[1, 2, 3, 4, 5]$ $[1, 2, 3, 0, 0] = 3$
compref $[1, 2, 3]$ $[9, 9, 9, 9, 9, 9] = 0$.

Wskazówka: Można korzystać ze standardowych funkcji listowych, w tym z funkcji zip, która z dwóch list tworzy listę par, np. zip $[1, 2]$['a', 'b', 'c'] $= [(1, \text{'a'}), (2, \text{'b'})]$.

Programowanie funkcyjne — kolokwium nr 1, 7.12.2022

Instrukcja: Każde zadanie należy przesłać na Pegaza w oddzielnym pliku: zadanie1.hs, zadanie2.hs i zadanie3.hs. Plików nie należy zipować. Rozwiązania muszą się poprawnie kompilować. W rozwiązaniach nie można korzystać z modułów innych niż standardowe; niedozwolone jest użycie polecenia import. Rozwiązania nie spełniające powyższych wymogów nie będą oceniane. Punktacja: 10 punktów za każde zadanie. Uwaga: korzystanie z internetu poza wyznaczonym czasem skutkuje automatycznym otrzymaniem 0 punktów.

Zadanie 1. Napisać funkcję o sygnaturze

`cztery :: [Int] -> Int`,

która dla podanej listy liczb całkowitych zwróci długość najdłuższego spójnego podciągu o sumie elementów podzielnej przez 4.

Zadanie 2. Napisać funkcję o sygnaturze

`wyniki :: Double -> [(String,Double)] -> [(String,String)]`,

która w pierwszym argumencie otrzymuje maksymalną liczbę punktów do uzyskania z testu, w drugim zaś otrzymuje listę par, które na pierwszej współrzędnej przechowują dane osób piszących test, a na drugiej liczbę uzyskanych punktów. Funkcja ta ma zwracać listę par zmodyfikowanych w następujący sposób:

- Jeśli na pierwszej współrzędnej na początku lub na końcu napisu znajdują się spacje, należy je usunąć (przed pierwszym znakiem, który nie jest spacją, **oraz** za ostatnim znakiem, który nie jest spacją; można założyć, że napisy nie składają się tylko ze spacji). Poza tym napis ma pozostać bez zmian.
- Wynik punktowy z drugiej współrzędnej ma zostać zmieniony w zależności od tego, w jaki przedział procentowy maksymalnego wyniku wpada:

  $(90\%, 100\%] \rightarrow$ `"5.0"`
  $(80\%, 90\%] \rightarrow$ `"4.5"`
  $(70\%, 80\%] \rightarrow$ `"4.0"`
  $(60\%, 70\%] \rightarrow$ `"3.5"`
  $(50\%, 60\%] \rightarrow$ `"3.0"`
  $[0\%, 50\%] \rightarrow$ `"2.0"`

  Jeśli wynik punktowy jest mniejszy od zera lub większy od maksymalnej liczby punktów, należy go zamienić na napis `"Nieprawidłowe dane"`.

Przykładowo, wywołanie

`wyniki 100.0 [(" Arnold ",60.4), ("Franek",75), (" Tim", 991.93)]`

powinno zwrócić

`[("Arnold","3.5"),("Franek","4.0"),("Tim","Nieprawidłowe dane")]`.

Zadanie 3. *Bluszcz skierowany* to struktura danych, która pozwala na: dołączenie elementu (`de`), odczytanie elementu ostatnio dołączonego (`oe`), usunięcie elementu ostatnio dołączonego (`ue`), podanie liczby elementów równych elementowi dołączonemu jako pierwszy (`le`) oraz zamianę całej struktury na listę (`bsk2l`), przy czym dowolne dwa elementy dołączone w następujących po sobie operacjach `de` muszą być sąsiadami na liście. Zamiana na listę powinna być wykonalna w czasie liniowym względem łącznej liczby elementów, natomiast pozostałe operacje — w czasie stałym. Zdefiniować typ `Bsk a`, służący do przechowywania elementów typu `a` w bluszczu skierowanym, oraz następujące funkcje, realizujące opisane wyżej operacje z odpowiednią złożonością:

```
de :: Bsk a -> a -> Bsk a
oe :: Bsk a -> a
ue :: Bsk a -> Bsk a
le :: Eq a => Bsk a -> Integer
bsk2l :: Bsk a -> [a]
```

Funkcje `oe`, `ue` i `le` nie są zdefiniowane dla bluszczu pustego.