Midterm FPRO, 2016 Meno a priezvisko: _____
obsahuje 4 príklady, spolu 6+5+(7+)+4= 22+ bodov skupina: bac-4k / mag-6k

1) **[6 bodov] Zoznamový -** V celom príklade je abeceda symbolov množina znakov ['a'..'z']. Príklad je o reťazcoch, teda typu String, čo je v Haskelli zoznam znakov. Teda "ahoj" a ['a','h','o','j'] sú identické hodnoty, preto s reťazcom pracujeme ako so zoznamom typu [Char].

- **[1 bod]** definujte funkciu **`unikatny::[String]->String`**, ktorá vráti ľubovoľný reťazec, ktorý sa **nenachádza** v neprázdnom vstupnom zozname neprázdnych reťazcov. Poznámka: vstupný zoznam môže mať aj jeden prvok...
- **[2 body]** definujte funkciu **`najdlhsie::[String]->[String]`**, ktorá vráti zoznam všetkých najdlhších reťazov (maximálnej dĺžky) zo vstupného zoznamu, na ich poradí nezáleží. Ak sa vám to podarí na jeden prechod vstupného zoznamu, **[+1bod]**.
- **[3 body]** definujte funkciu `najkratsi::[String]->String`, ktorá vráti **ľubovoľný** najkratší neprázdny reťazec, ktorý sa nenachádza vo vstupnom zozname reťazcov - nie je jednoznačný.
- **Príklady (ilustrujú zadania, ale nepredpisujú niť, čo nie je v zadaní špecifinované):**

```
unikatny ["a", "ab", "b"] = "aabb"
najdlhsie ["a", "ab", "b", "aa"] = ["ab","aa"]
najdlhsie ["a", "ab", "b", "aa","abc"] = ["abc"]
najkratsi ["a", "ab", "b", "aa","abc"] = "c"
najkratsi $ map(\c->[c])['a'..'z'] = "aa"
```

ďalšie riešenia príkladu 1 píšte na tento list (aj z druhej strany) alebo si vypýtajte ďalší list

2) [5 bodov] Stromový ï Binárny vyhÕadávací strom je definovaný

```
data Tree a = Empty | Node a (Tree a) (Tree a) deriving (Show, Read)
```

- **[2 body]** definujte funkciu **`minim::(Ord a)=>Tree a->a`**, ktorá nájde a vráti minimimum v neprázdnom binárnom vyhÕadávacom strome. Predpokladajte, ģe vstupný strom spĒa vlastnosŠ binárneho vyhÕadávacieho stromu ï v tom prípade nepotrebujete predpoklad **`(Ord a)`…**
- **[3 body]** definujte funkciu **`delete::Ord a=>a->Tree a->Tree a`** , ktorá vyhodí prvok z binárneho vyhÕadávacieho stromu. OpäŠpredpokladajte, ģe vstupný strom spĒa vlastnosŠbinárneho vyhÕadávacieho stromu. Ak sa hodnota v strome nenachádza, strom sa nezmení.
- **Príklady, ak `t = (Node 2 (Node 1 Empty Empty) (Node 3 Empty Empty))`:**

```
minim t = 1
delete 1 t = Node 2 Empty (Node 3 Empty Empty)
delete 2 t = Node 3 (Node 1 Empty Empty) Empty
delete 3 t = Node 2 (Node 1 Empty Empty) Empty
```

ï ï ï ï ï ï ï ï ï ï rieģenia príkladu 2 píģte na tento list (aj z druhej strany) alebo si vypýtajte Ņalģí list

3) [7+ bodov] Lambdový ï λ-termy sme definovali

```
type Var = String
data LExp = LAMBDA Var LExp | ID Var | APP LExp LExp  deriving(Eq)
```

- **[2 body]** λ-term je uzavretý, ak neobsahuje voľné premenné, teda žiaden výskyt žiadnej premennej nie je voľný. Definujte funkciu **`uzavrety::LExp->Bool`**, ktorá to testuje.
- **[3 body]** α-konverzia hovorí o premenovaní premennej v λ-abstrakcii a následne vždetkých viazaných výskytov tejto premennej v tele abstrakcie. α-konverzia indukuje α-ekvivalenciu. Definujte funkciu **`alpha::LExp->LExp->Bool`**, ktorá zistí, či dva **uzavreté** λ-termy sú α-ekvivalentné.
- **[2+ bod]** β-redukcia aplikuje λ-abstrakciu na argument v terme obsahujúcom redex (t.j. nie je v β-normálnej forme). Zamyslite sa nad nasledujúcimi tvrdeniami. Ak neplatia, uveďte kontrapríklad, ak platia, oznaľte, že platia, dôkaz netreba... Kontrapríklad, ak tvrdenie platí, sa nehodnotí :-)

  - dva α-ekvivalentné λ-termy sú buďto oba v β-normálnej forme, alebo oba obsahujú β-redex.

  platí [0.5 bodu]
  neplatí(kontrapríklad) [1 bod]:

  - ak λ-term nie je v β-normálnej forme (obsahuje β-redex), potom β-redukcia **zmení** tento λ-term. Inými slovami, λ-termy pred a po β-redukcii sú vždy rôzne,

  platí [0.5 bodu]
  neplatí(kontrapríklad) [1 bod]:

  - dva λ-termy pred a po β-redukcii sú α-ekvivalentné,

  platí [0.5 bodu]
  neplatí(kontrapríklad) [1 bod]:

  - β-redukcia zachováva α-ekvivalenciu - ak dva termy, ktoré nie sú v normálnej forme (s nejakým β-redexom), sú α-ekvivalentné λ-termy, potom tieto λ-termy po β-redukcii sú tiež α-ekvivalentné

  platí [0.5 bodu]
  neplatí(kontrapríklad) [1 bod]:

- **Príklady:**

```
uzavrety (LAMBDA "x" (LAMBDA "y" (APP (ID "y") (ID "x")))) = True
uzavrety (LAMBDA "x" (LAMBDA "y" (APP (ID "z") (ID "x")))) = False
alpha      (LAMBDA "w" (LAMBDA "z" (APP (ID "z") (ID "w"))))
           (LAMBDA "x" (LAMBDA "y" (APP (ID "y") (ID "x")))) = True
alpha      (LAMBDA "w" (LAMBDA "z" (APP (ID "w") (ID "z"))))
           (LAMBDA "x" (LAMBDA "y" (APP (ID "y") (ID "x")))) = False
```

ï ï ï ï ï ï ï ï ï ï ï rieženia príkladu 3 píšte na tento list alebo si vypýtajte ďalší list

4) [4 body = 1 bod za kaġdú správnu odpoveĎ] kvízový

U kaġdého tvrdenia sa zamyslite, ļi platí. Ak neplatí, napíġte kontrapríklad. Ak platí, napíġte pozitívny príklad ilustrujúci jedno pouġitie tvrdenia. Príklady musia byŠtypovo správne, inak sa nehodnotia.

Pozor:

- pozitívny príklad nie je dôkaz, ani si to nemyslite, ġe ste tvrdenie dokázali,
- ak tvrdenie neplatí, vaġ pozitívny príklad sa nehodnotí, podobne aj naopak :-)

1) (map f) . (drop n) = (drop n) . (map f)

platí, príklad:

neplatí, kontrapríklad:

2) (take n) . (filter p) = (filter p) . (take n)

platí, príklad:

neplatí, kontrapríklad:

3) (map f) . concat = concat . (map (map f))

platí, príklad:

neplatí, kontrapríklad:

4) concat . (map f) = foldr (\x -> \y -> f x ++ y) []

platí, príklad:

neplatí, kontrapríklad:

ï ï ï ï ï ï ï rieġenia príkladu 4 píġte na tento list, aj zozadu, alebo si vypýtajte Ďalġí list