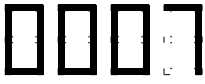


Jméno:

UČO:

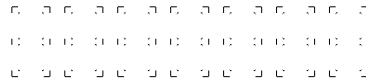
Souřadnice:



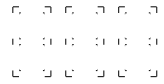
list



učo



body



Oblast strojově snímaných informací. Svě učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte.

0 1 2 3 4 5 6 7 8 9

Test minimálních dovedností

Příklady v této části písemky ověřují základní znalosti a dovednosti. Vyřešení všech příkladů v této části je nutné pro úspěšné ukončení předmětu. Tato část písemky se neboduje. Bude prominuta nejvýše jedna drobná chyba (prohození pořadí argumentů u funkce bez zadaného typu, evidentní typo, apod.).

1. Uveďte, na jakou hodnotu se vyhodnotí následující výraz v jazyce Haskell:

```
map fst (zip (replicate 3 ":") ["A","B","C"])) ~>*
```

2. Uvažme funkci $fn :: [Int] \rightarrow [Int]$, která pro zadaný seznam vrátí seznam, ve kterém je prvek na dané pozici nahrazen sumou suffixu seznamu od dané pozice nevčetně.

Příklady vyhodnocení:

```
fn [1,1,1] ~>* [2,1,0]
```

```
fn [3,2,1] ~>* [3,1,0]
```

```
fn [1] ~>* [0]
```

```
fn [] ~>* []
```

a) Definujte funkci `fn` bez použití explicitní rekurze, tj. s použitím funkcí definovaných v modulu `Prelude` (`head`, `tail`, `last`, `init`, `reverse`, `length`, `take`, `drop`, `map`, `filter`, `zip`, `zipWith`, `(:)`, `[]`, `(+)`, `(*)`, `sum`, `...`), případně s využitím intensionálního způsobu konstrukce seznamu.

```
fn list =
```

b) Stejnou funkci definujte též s využitím explicitní rekurze na zadaném seznamu. Pro tento účel si smíte zadefinovat pomocnou funkci a přenést rekurzi na zadaném seznamu do této pomocné funkce, nesmíte se však explicitní rekurzi vyhnout.

Jméno:

UČO:

Souřadnice:

0007

list

2

učo

body

Oblast strojově snímaných informací. Svě učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte.

0123456789

3. Uveďte typ výrazu $\backslash y \rightarrow (\text{snd } y) ((\text{not.fst}) y)$. Typy použitých funkcí jsou:

$\text{snd} :: (a, b) \rightarrow b$

$\text{fst} :: (a, b) \rightarrow a$

$\text{not} :: \text{Bool} \rightarrow \text{Bool}$

$\backslash y \rightarrow (\text{snd } y) ((\text{not.fst}) y) ::$

4. Mějme uživatelem definován následující typ:

```
data TreTree a = Node (a,a) (TreTree a, TreTree a, TreTree a) | Leaf a
```

a) Uveďte libovolnou platnou hodnotu typu `TreTree Char`.

b) Uveďte alespoň jeden úplný typ definovaný s nově zavedeným typovým konstruktorem.

c) Napište funkci `monoLeaf :: Int → TreTree Int → Bool`, která rozhodne, zda všechny listy stromu (2. argument) obsahují zadanou hodnotu (1.argument).

Jméno:

UČO:

Souřadnice:

0007

list

3

učo

body

Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte.

0123456789

Pokročilá část

Body, které získáte v této části, se sečtou s body získanými za domácí úlohy, celková suma určí Vaši známku. Tato část písemky Vám bude opravena, pouze pokud úspěšně vyřešíte první část písemky.

5. [4 body] Nerekurzivním způsobem definujte funkci `decodeRLE :: [(Int, Char)] -> IO ()`. Která pro zadaný RLE kód vytiskne dekódovaný řetězec na terminál. Může se vám hodit funkce `replicate :: Int -> a -> [a]`.

Příklady vyhodnocení:

`decodeRLE [(2, '0'), (1, '7')]` vytiskne "007"

`decodeRLE [(1, 'H'), (3, 'u')]` vytiskne "Huuu"

`decodeRLE []` nevytiskne nic (vytiskne prázdný řetězec)

Jméno:

UČO:

Souřadnice:

0007

list

4

učo

body

Oblast strojově snímaných informací. Svě učo a číslo listu vyplňte
zleva dle vzoru číslic. Jinak do této oblasti nezasahujte.

0123456789

6. [2 body] Uvažme následující uživatelem definovaný typ `ITree a` a funkci `kmIT`.

```
data ITree a = ILeaf | INode a (ITree a) (ITree a)
```

```
kmIT :: a -> (b -> a -> a -> a) -> (ITree b) -> a
```

```
kmIT lf nf (ILeaf) = lf
```

```
kmIT lf nf (INode v x y) = nf v (kmIT lf nf x) (kmIT lf nf y)
```

Doplňte definici funkce `sumITree` tak, aby její následná aplikace na hodnotu typu `ITree Int`, vrátila sumu hodnot uložených ve vnitřních uzlech zadaného stromu.

```
sumITree = kmIT
```

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

7. [4 body] V jazyce **PROLOG** naprogramujte predikát `take(L,N,R)`, který je pravdivý, pokud seznam `R` je prefixem délky `N` seznamu `L`. Předpokládejte použití predikátu `len` pouze v `(+,+,+)` módu.