1. [povinné][Haskell] Naprogramujte funkci `myDrop :: Int → [a] → [a]`, která se chová stejně jako její obdoba ze standardní knihovny `drop`. Volání `myDrop n xs` vrátí seznam, který je rovný seznamu `xs` po vynechání `n` počátečních prvků. Pokud je číslo `n` větší než délka seznamu `xs`, výsledkem je prázdný seznam. Pro záporné hodnoty prvního argumentu funkce nemusí být definovaná nebo může cyklit. Můžete využít libovolné konstrukce jazyka Haskell, nesmíte však použít knihovní funkci `drop`.

Příklady vyhodnocení:

```
myDrop 0 ['a', 'b', 'c', 'd'] ⇝* ['a', 'b', 'c', 'd']
myDrop 2 ['a', 'b', 'c', 'd'] ⇝* ['c', 'd']
myDrop 4 ['a', 'b', 'c'] ⇝* []
myDrop 6 [] ⇝* []
```

2. [povinné][Haskell] Uveďte typ výrazu `takeWhile even`. Typy elementárních výrazů jsou:

```
takeWhile :: (a → Bool) → [a] → [a]
even :: Integral a ⇒ a → Bool
```

---

3. [povinné][Haskell] Mějme dány datové typy `Point` a `LPicture`, jejichž hodnoty slouží k popisu dvojrozměrného vektorového obrázku složeného z jednoduchých čar. (Každá čára je popsána dvěma krajními body, každý bod je určen souřadnicemi x a y, tj. dvěma celými čísly.) Obrázek je tvořen buď samostatnou čárou, nebo kompozicí dvou obrázků.

```
data Point = P Int Int
data LPicture = Line Point Point | Comp LPicture LPicture
```

a) Uveďte libovolnou platnou hodnotu typu `LPicture`, při jejíž konstrukci je použit datový konstruktor `P` typu `Point`.

b) Napište funkci `moveBy :: Point → LPicture → LPicture`, která posune souřadnice všech bodů použitých pro popis obrázku o hodnoty zadané bodem v prvním argumentu funkce.

4. [povinné][Prolog] Zakreslete kompletní SLD strom pro dotaz `?- r(X, Y).` a databázi faktů níže. Pro úspěšné větve zapište výslednou substituci, kterou by interpretr vypsal na obrazovku.

```
f(c).
f(a).
r(X, Y) :- f(X), d(X, Y).
d(c, Y) :- f(Y).
```

5. [povinné][Prolog] Naprogramujte predikát `increasingList(+List)`, který uspěje právě tehdy, pokud prvky seznamu `List` tvoří rostoucí posloupnost čísel. Můžete předpokládat, že `List` je plně instanciovaný seznam čísel.

Příklady vyhodnocení:

```
?- increasingList([1,2,3]).
true.

?- increasingList([2,3,1]).
false.

?- increasingList([]).
true.

?- increasingList([42]).
true.
```

Příklady v této části písemky jsou bodované. Body, které získáte spolu s body za domácí úlohy a aktivitu na cvičeních, rozhodnou o Vaší známce. Tato část písemky Vám bude opravena, pouze pokud úspěšně vyřešíte základní část.

---

6. [5 bodů][Haskell] Uveďte typ výrazu `\x y → [(x y), "pickle", "rick"]`.
Typy elementárních výrazů jsou:

```
"pickle" :: String
"rick" :: String
```

7. [10 bodů][Haskell] Naprogramujte **pomocí foldovacích funkcí** funkci `reverseList` typu `[a] → [a]`, která vrátí zadaný seznam pozpátku. Nesmíte využít knihovní funkci `reverse`, můžete ale využít libovolné jiné knihovní funkce či konstrukce jazyka Haskell.

Vaše řešení musí být ve tvaru `reverseList xs = fold{l,r,l1,r1}...` (případně je možné vynechat argument).

Příklady vyhodnocení:

```
reverseList [] = []
reverseList [1,2,3] = [3,2,1]
reverseList "abcd" = "dcba"
```

8. [10 bodů][Haskell] Uvažte následující typ pro reprezentaci binárních stromů s hodnotami v listech v jazyce Haskell:

```
data LeafTree a = Fork (LeafTree a) (LeafTree a)
                | Leaf a
```

Definujte instanci typové třídy `Eq` pro datový typ `LeafTree a`. Jinak řečeno, zajistěte, že pro libovolný porovnatelný datový typ `a` bude možné porovnávat hodnoty typu `LeafTree a` pomocí funkce `(==)`. Porovnávání by mělo testovat, zda jsou stromy identické. Nesmíte nijak měnit definici datového typu `LeafTree a`, tedy ani k ní přidávat klauzule `deriving`. Pro připomenutí: minimální kompletní implementace třídy `Eq` obsahuje pouze funkci `(==)`.

9. [10 bodů][Prolog] Naprogramujte predikát `firstWith42(+ListOfLists, ?X)`, který pro zadaný seznam seznamů `ListOfLists` uspěje právě tehdy, pokud `X` je *první* seznam v tomto seznamu seznamů, který obsahuje číslo 42. Můžete předpokládat, že `ListOfLists` je seznam a že každý jeho prvek je plně instanciovaný seznam čísel. Váš predikát `firstWith42` musí fungovat i pro neinstanciované `X`; v takovém případě se do proměnné `X` unifikuje první takový seznam. Můžete použít libovolné konstrukce a knihovní funkce jazyka Prolog; zejména by se vám mohl hodit predikát `member(?Element, ?List)`.

Příklady vyhodnocení:

```
?- firstWith42([[1], [1,42], [2], [3,42]], [1,42]).
true.

?- firstWith42([[1], [1,42], [2], [3,42]], [3,42]).
false.

?- firstWith42([[1], [1,42], [2], [3,42]], X).
X = [1, 42].
```