Faculty of Informatics
Masaryk University

Cvičení k předmětu IB015
Neimperativní programování

poslední modifikace 7. prosince 2013

Tento text vznikl přepsáním a restrukturalizací cvičení k dnes již neexistujícímu předmětu *IB015 Úvod do funkcionálního programování*. Primárním autorem příkladů pro funkcionální programování je RNDr. Libor Škarvada a nesčetní cvičící bývalého předmětu IB015, kteří znění příkladů redigovali.

Cvičení 1

1.1 Seznamte se s ovládáním programu interpretru jazyka Haskell (`ghci`, případně `hugs`).

1.2 S použitím interpretru jazyka Haskell porovnejte vyhodnocení následujících dvojic výrazů a rozdíl vysvětlete.

a) `5+9*3` versus `(5+9)*3`

b) `2^2^2==(2^2)^2` versus `3^3^3==(3^3)^3`

c) `3+3+3` versus `3==3==3`

d) `(3==3)==3` versus `(4==4)==(4==4)`

1.3 S využitím interního příkazu `:info` interpretru `ghci` zjistěte prioritu a směr vyhodnocování následujících operací:

```
., !!, ^, *, /, `div`, `mod`, +, -, :, ++, ==, /=, >, <, &&, ||
```

1.4 Přepište infixové zápisy výrazů do syntakticky správných prefixově zapsaných výrazů a naopak:

a) `4^(7 `mod` 5)`

b) `max 3 ((+) 2 3)`

1.5 Výraz `((3+4)/2) * ((3+4)/2-3) * ((3+4)/2-4)`

a) upravte s využitím syntaktické konstrukce pro lokální definici (`let ... in`) tak, aby se v něm neopakovaly stejné složené podvýrazy.

b) upravte stejně, ovšem s využitím globálních definic (uložených v externím souboru).

1.6 Vysvětlete, co je chybné na následujících podmíněných výrazech, a výrazy vhodným způsobem upravte.

a) `if 3 `mod` 2 then False else True`

b) `if 0<3 && odd 6 then 1 else "chyba"`

c) `(if even 8 then (&&)) (0>7) True`

1.7 Definujte funkci `teleAsele`, která se chová stejně jako funkce logické konjunkce, tak, abyste v definici

a) využili podmíněný výraz.

b) nepoužili podmíněný výraz.

1.8 Upravte a doplňte následující zdrojový kód tak, aby program vyžadoval a načetl postupně tři celá čísla a o nich určoval, zda mohou být délkami hran trojúhelníku. Hotový program přeložte do samostatného spustitelného souboru a otestujte.

```
main = do putStr "Dej mi jedno číslo: "
          x <- getLine
          print ((+) 1 (read x::Int))
```

Cvičení 2

2.1 Definujte rekurzivní funkci pro výpočet faktoriálu.

2.2 Upravte následující kód tak, aby funkce pro záporná čísla necyklila, ale skončila s chybovou hláškou. Použijte k tomu funkci `error :: String -> a`.

```
na :: Double -> Int -> Double
_ `na` 0 = 1
z `na` n = z * (z `na` (n-1))
```

2.3 Definujte v Haskellu funkci `dfct` (v kombinatorice někdy značenou !!), kde

$0!! = 1$,
$(2n)!! = 2.4.....(2n)$,
$(2n+1)!! = 1.3....(2n+1)$

2.4 Co počítá následující funkce? Jak se chová na argumentech, kterými jsou nezáporná čísla? Jak se chová na záporných argumentech?

```
sq 0 = 0
sq n = sq (n-1) + 2*n - 1
```

2.5 Definujte funkci `posledni :: [a] -> a`, která vrátí poslední prvek neprázdného seznamu. Nesmíte použít funkci `last`.

2.6 Definujte funkci `zacatek :: [a] -> [a]`, která pro neprázdný seznam vrátí tentýž seznam bez posledního prvku. Nesmíte použít funkci `init`.

2.7 S využitím funkce `map` a knihovní funkce `toUpper :: Char -> Char` z modulu `Data.Char` (tj. je třeba použít `import Data.Char;`) definujte novou funkci `toUpperStr`, která převádí řetězec písmen na řetězec velkých písmen, tj.

`toUpperStr "bob"` $\rightsquigarrow^*$ `"BOB"`.

2.8 Popište, jak se chová následující funkce a pokuste se ji definovat kratším zápisem a efektivněji.

```
s2m 0 = [0]
s2m n = s2m (n-1) ++ [ last (s2m(n-1)) + 2*n - 1 ]
```

Bonus: Dokažte, že je vaše nová definice ekvivalentní.

2.9 Definujte funkci `odmocniny :: [Double] -> [Double]`, která ze zadaného seznamu vybere kladná čísla a ta odmocní. (Využijte mimo jiné funkce `filter`, `(>0)` a `sqrt`.)

2.10 Funkci `zip :: [a] -> [b] -> [(a,b)]`, lze definovat následovně:

```
zip (x:s) (y:t) = (x,y) : zip s t
zip _ _ = []
```

a) Které dvojice parametrů vyhovují prvnímu řádku definice?

b) Přepište definici tak, aby první klausule definice (první řádek) byla použita jako poslední klausule definice.

2.11 Definujte funkci `zip3 :: [a] -> [b] -> [c] -> [(a,b,c)]`.

2.12 Funkce `unzip :: [(a,b)] -> ([a],[b])` může být definována následovně:

```
unzip [] = ([],[])
unzip ((x,y):s) = (x:u,y:v) where (u,v) = unzip s
```

Definujte analogicky funkce `unzip3`, `unzip4`, ...

2.13 Jaká je hodnota následujících výrazů?

a) `zipWith (:) "MF"["axipes","ík"]`

b) `zipWith (^) [1..5] [1..5]`

c) `let fibs = [0,1,1,2,3,5,8,13] in zipWith (+) fibs (tail fibs)`

d) `let fibs = [0,1,1,2,3,5] in zipWith (/) (tail (tail fibs)) (tail fibs)`

2.14 Vymyslete (vzpomeňte si) na další funkce pracující se seznamy, pojmenujte je v Haskellu nerezervovaným slovem, definujte je a vyzkoušejte svoji definici v interpretru jazyka Haskell.

Cvičení 3

3.1 Následující výrazy použijte v lokální definici a vyhodnoťte v interpretru jazyka Haskell na vhodných parametrech. Po úspěšné aplikaci výrazy upravujte tak, abyste se při jejich definici vyhnuli použití lambda abstrakce a formálních parametrů.

a) `\x -> 3*x`

b) `\x -> x^3`

c) `\x -> 3 + 60 ‘div‘ x^2 >0`

d) `\x -> [x]`

e) `\s -> "("++ s ++ ")"`

f) `\x -> 0 < 35 - 3 * 2^x`

g) `\x y -> x^y`

h) `\x y -> y^x`

i) `\x y -> 2 * x + y`

3.2 Co vyjadřuje výraz `min 6`? Napište ekvivalentní výraz pomocí `if`.

3.3 Definujte unární funkci `nebo` pro realizaci logické disjunkce a pomocí modifikátorů `curry` a `uncurry` definujte ekvivalenci mezi vámi definovanou funkcí `nebo` a předdefinovanou funkcí `||`.

3.4 Analogicky k funkcím `curry` a `uncurry` definujte funkce

a) `curry3 :: ((a,b,c) -> d) -> a -> b -> c -> d`

b) `uncurry3 :: (a -> b -> c -> d) -> (a,b,c) -> d`

3.5 Zaveďme `distr f g x = f x (g x)` .

a) Vyjádřete funkci `distr (curry id) id` pomocí lambda abstrakce.

b) Co dělá funkce `pair = uncurry (distr . ((.)(curry id)))`

3.6 Uvažme funkci `negp :: (a -> Bool) -> a -> Bool`, která neguje výsledek unárních funkcí typu `a -> Bool` (tzv. predikátů).

a) Vysvětlete, vlastními slovy, co dělá funkce `negp`.

b) Definujte funkci `negp`.

c) Definujte funkci `negp` bez použití formálních parametrů.

3.7 Určete typy seznamů:

a) `["a","b","c"]`

b) `['a','b','c']`

c) `"abc"`

d) `[(True,()),(False,())]`

e) `[map toUpper "abc", map toLower "XYZ"]`

f) `[(&&),(||)]`

g) `[]`

h) `[[]]`

i) `[[],[""]]`

j) `head . tail`

k) `tail . head`

3.8 Určete typy funkcí:

a) `swap (x,y) = (y,x)`

b) `cadr = head . tail`

c) `caar = head . head`

d) `twice f = f . f`

e) `comp12 g h x y = g (h x y)`

Nepovinné domácí cvičení. Přepište následující definice funkcí tak, abyste v jejich definici nepoužili lambda abstrakci a formální parametry (tj. chce se pointfree definice).

a) `f x y = y`

b) `h x y = q y . q x`

Nepovinné domácí cvičení. Zjistěte, co dělají následující funkce

a) `h1 = (.(,)) . (.) . (,)`

b) `h2 = ((,).) . (,)`

Cvičení 4

4.1 Uvažme funkci: `foldr (.) id`

a) Jaký je význam uvažované funkce?

b) Jaký je její typ?

c) Uveďte příklad částečné aplikace této funkce na jeden argument.

d) Uveďte příklad úplné aplikace této funkce na kompletní seznam argumentů.

4.2 Předpokládejme, že funkce `compose` skládá všechny funkce ze seznamu funkcí, tj. `compose [`f_1`, ..., `f_n`] = `f_1` . ... . `f_n.

a) Definujte funkci `compose` s využitím akumulátorových funkcí.

b) Jaký je typ funkce `compose`? (Odvoďte bez použití interpretru a poté ověřte.)

4.3 Definujte funkci `subtractlist`, která odečte druhý a všechny další prvky *neprázdného* seznamu od jeho prvního prvku, tj. `subtractlist [`x_1`, ... , `x_n`] = `x_1` - `x_2` - ... - `x_n.

4.4 Jaký je význam a typ funkce `foldl (flip (:)) []`?

4.5 Jaký je význam a typ funkce `foldr (:)`?

4.6 Mějme následující definici:

```
data Teleso = Kvadr Float Float Float -- a,b,c
            | Valec Float Float       -- r,v
            | Kuzel Float Float       -- r,v
            | Koule Float             -- r
```

a) Jaké hodnoty má typ `Teleso`?

b) Kolik je v definici použito datových konstruktorů a které to jsou?

c) Kolik je v definici použito typových konstruktorů a které to jsou?

d) Definujte funkce `objem` a `povrch`, které pro hodnoty uvedeného typu počítají požadované.

e) Rozšiřte uvedený datový typ o další konstruktory a upravte odpovídajícím způsobem vámi definované funkce.

4.7 S využitím typového konstruktoru `Maybe` definujte funkci `divlist`, která podělí dva seznamy „po složkách“, tj. `divlist [`x_1`, ... , `x_n`] [`y_1`, ... , `y_n`]` $\rightsquigarrow^*$ `[`x_1`/`y_1`,...,`x_n`/`y_n`]`.

4.8 Do interpretru zapište nejprve výraz `"**\n**\n"` a poté výraz `putStr "**\n**\n"`. Rozdíl chování interpretru vysvětlete.

4.9 Vysvětlete význam a rizika rekurzivního použití volání funkce `main` v následujícím programu.

```
main :: IO ()
main = do putStr "řetězec: "
          s <- getLine
          if null s then putStrLn "pa pa"
                    else do putStrLn (reverse s)
                            main
```

4.10 Napište program, který vyzve uživatele, aby zadal jméno souboru, poté ověří, že zadaný soubor existuje, a pokud ano, vypíše jeho obsah na obrazovku, pokud ne, informuje o tom uživatele.

a) Úkol řešte s využitím `doesFileExist` z modulu `System.Directory`

b) Úkol řešte s využitím výjimek (funkce `catch`).

4.11 Uvažme následující program:

```
import Data.Char
main = getLine >>= putStr . filter isAlpha
```

a) Co program dělá?

b) Přepište program do `do` notace.

4.12 Následující funkci přepište do tvaru, ve kterém nepoužijete konstrukci `do`, také určete typ funkce.

```
query otazka = do putStr otazka
                  odpoved <- getLine
                  return (odpoved == "ano")
```

4.13 Upravte program `hadej.ps` tak, aby parametry funkce `hadej` četl z příkazové řádky. Program je možné stáhnout z následující URL:

`http://www.fi.muni.cz/~xbarnat/IB015/hadej.hs`

4.14 Funkci `query` z předchozího příkladu modifikujte tak, aby:

a) Rozlišovala kladné i záporné odpovědi a při nekorektní nebo nerozpoznané odpovědi otázku opakovala.

b) Akceptovala odpovědi s malými i velkými písmeny, interpunkcí, případně ve více jazycích.

4.15 Definujte akci `getInt :: IO Int`, která ze standardního vstupu načte celé číslo. Využijte knihovní funkci `read :: (Read a) => String -> a`.

4.16 Vymyslete a naprogramujte několik triviálních prográmků manipulujících se soubory. Definice alternativně přepište s a bez pomoci syntaktické konstrukce `do`.

Cvičení 5

5.1 Intensionálním způsobem zapište následující výrazy:

a) `map f s`

b) `filter p s`

c) `map f (filter p s)`

d) `repeat x`

e) `replicate n x`

5.2 Je možné zapsat intensionálním způsobem prázdný seznam? Pokud ano, jak, pokud ne, proč?

5.3 Pomocí funkce `iterate` vyjádřete nekonečné seznamy:

a) Rostoucí seznam všech mocnin čísla 2.

b) Rostoucí seznam všech sudých mocnin čísla 3.

c) Rostoucí seznam všech lichých mocnin čísla 3.

d) Seznam řetězců `["","*","**","***","****",...]`.

5.4 Pomocí rekurzivní definice a funkce `zipWith` vyjádřete Fibonacciho posloupnost, tj. seznam čísel `[0,1,1,2,3,5,8,13,21,34,...]`.

5.5 Definujte Fibonacciho posloupnost bez použití funkce `zipWith`.

5.6 Uvažme seznam definovaný následovně:

```
pt :: [[Integer]]
pt = iterate (\r -> zipWith (+) ([0]++r) (r++[0])) [1]
```

a) Vypište jeho prvních 15 prvků.

b) Co seznam vyjadřuje?

c) Vyhodnoťte `take 7 pt`.

d) Vyhodnoťte `show (take 7 pt)`.

e) Vyhodnoťte `map show (take 7 pt)`.

f) Vyhodnoťte `(unlines . map show) (take 7 pt)`.

g) Vyhodnoťte `(putStr . unlines . map show) (take 7 pt)`.

5.7 Z níže uvedené URL stáhněte program `pt.hs`, spusťe ho a pochopte, jak funguje.

`http://www.fi.muni.cz/~xbarnat/IB015/pt.hs`

Cvičení 6

6.1 Dokažte, že následující funkce zjišťuje počet prvků (tj. délku) svého argumentu – seznamu.

```
length :: [a] -> Int
length []   = 0
length (_:s) = 1 + length s
```

6.2 Uvažme následující rekurzivní datový typ:

```
data Nat = Zero | Succ Nat deriving Show
```

a) Jaké hodnoty má typ `Nat`?

b) Jaký význam má dovětek `deriving Show`?

c) Redefinujte způsob zobrazení hodnot typu `Nat`.

6.3 Pro datový typ `Nat` z předchozího příkladu definujte funkci `nfold`, která je tzv. katamorfismem na typu `Nat` (něco jako akumulační funkce na struktuře, jež je schovaná za každou jednou hodnotou typu `Nat`).

```
nfold :: (a -> a) -> a -> Nat -> a
```

Příklady zamýšleného použití funkce `nfold`:

- Funkce `nfold (Succ . Succ) Zero :: Nat -> Nat` „zdvojnásobuje“ hodnotu typu `Nat`.
- Funkce `nfold (1+) 0 :: Nat -> Int` převádí hodnotu typu `Nat` do celých čísel typu `Int`.

6.4 Uvažme následující definici typu `Expr`:

```
data Expr = Con Float
          | Add Expr Expr | Sub Expr Expr
          | Mul Expr Expr | Div Expr Expr
```

a) Uveďtě výraz typu `Expr`, který představuje hodnotu 3.14.

b) Definujte funkci `eval :: Expr -> Float`, která vrátí hodnotu daného výrazu.

c) Ošetřete korektně dělení nulou pomocí funkce `evaluate :: Expr -> Maybe Float`.

6.5 Uvažme následujcí rekurzivní typ:

```
data BinTree a = Empty | Node a (BinTree a) (BinTree a)
```

a) Nakreslete všechny tříuzlové stromy typu `BinTree ()` a zapište je pomocí datových konstruktorů `Node` a `Empty`.

b) Kolik existuje stromů typu `BinTree ()` s 0,1,2,3,4 nebo 5 uzly?

c) Kolik existuje stromů typu `BinTree Bool` s 0,1,2,3,4 nebo 5 uzly?

d) Definujte funkci `size :: BinTree a -> Int`, která určí počet uzlů stromu.

6.6 Pro následující rekurzivní typ

```
data BinTree a = Empty | Node a (BinTree a) (BinTree a)
```

označíme *výškou stromu* počet uzlů na cestě z kořene do nejvzdálenějšího listu.

a) Definujte funkci `fulltree :: Int -> a -> BinTree a`, která pro volání `fulltree n v` vytvoří binární strom výšky `n`, ve kterém jsou všechny větve stejně dlouhé a všechny uzly ohodnocené hodnotou `v`.

b) Definujte funkci `height :: BinTree a -> Int`, která určí výšku stromu.

c) Definujte funkci `treezip :: BinTree a -> BinTree b -> BinTree (a,b)` jako analogii seznamové funkce `zip`.

6.7 Uvažme následující rekurzivní typ:

```
data BinTree a = Empty | Node a (BinTree a) (BinTree a)
```

a) Definujte funkci `treerepeat :: a -> BinTree a` jako analogii seznamové funkce `repeat`.

b) Pomocí funkce `treerepeat` vyjádřete nekonečný binární strom `niltree`, který má v každém uzlu prázdný seznam.

c) Definujte funkci `treeiterate :: (a->a) -> (a->a) -> a -> BinTree a` jako analogii seznamové funkce `iterate`.

Cvičení 7

7.1 Pro SEMESTR Podzim 2013: Dokončete příklady, které jste nestihli v předchozích cvičeních.

Cvičení 8

8.1 S využitím různých definic Fibonacciho posloupnosti z předchozích cvičení definujte funkci `fib :: Integer -> Integer`, která pro hodnotu n vrátí n-té Fibonacciho číslo. Porovnejte výkonnost výpočtu funkce `fib` pro různé definice posloupnosti a vysvětlete rozdíl.

8.2 Vysvětlete, co je to typová třída, a jak může programátorovi použití typové třídy pomoci (ušetřit) při tvorbě a definici vlastních typů.

8.3 Deklarujte typ `BinTree a` z předchozího cvičení jako instanci typové třídy `Eq`. Co vám tato instanciace „programátorsky“ přináší pro práci s tímto typem?

8.4 Deklarujte typ `Maybe` jako instanci typové třídy `Ord`. Za jakých podmínek může být typ `Maybe a` instancí třídy `Ord`?

8.5 Pochopte a vysvětlete následující definici typu `SearchTree a b`.

```
data BinTree a = Empty | Node a (BinTree a) (BinTree a)
type Ord a => SearchTree a b = BinTree (a,b)
```

8.6 Uvažme datové typy z předchozího příkladu.

a) Pro datový typ `BinTree a` definujte funkci `leftmostval`, která vrací hodnotu uloženou v nejlevějším uzlu daného stromu.

b) Jaký je nejobecnější typ funkce `leftmostval`?

c) Jaký je nejobecnější typ funkce `leftmostval`, pokud požadujeme, aby funkce pracovala pouze s typem `SearchTree a b`?

Cvičení 9

9.1 Pro SEMESTR Podzim 2013: Volné konzultace před vnitrosemestrální písemkou.

Cvičení 10

10.1 Spusťe a ukončete interpret jazyka Prolog.

10.2 Namodelujte osoby a rodičovské vztahy relacemi `dcera/2` a `syn/2` v rodině Simpsonových.

a) Naprogramujte predikát `sestra/2`, který se vyhodnotí na pravda, pokud osoba zadaná jako druhý parametr je sestrou osoby zadané jako první parametr.

b) Zformulujte dotaz, jehož úplným vyhodnocením se dozvíte všechny Bartovy sestry.

c) Podobně naprogramujte predikáty `bratr/2`, `deti/2`, `rodice/2`.

10.3 Uvažme následující databázi faktů:

```
r(a,b).
r(a,c).
r(b,d).

f1(a).
f1(X) :- f1(Y), r(Y,X).

f2(X) :- f2(Y), r(Y,X).
f2(a).

g1(a).
g1(X) :- r(Y,X), g1(Y).

g2(X) :- r(Y,X), g2(Y).
g2(a).
```

Zdůvodněte chování interpretru pro následující dotazy:

a) `?- f1(X).`

b) `?- f2(X).`

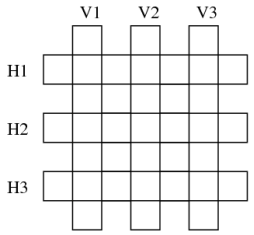
c) `?- g1(X).`

d) `?- g2(X).`

10.4 V PROLOGu implementujte predikát `fact/2`, který se při dotazu `fact(m,n)` vyhodnotí na `true .`, pokud `m>0` a `n = m!`, v ostatních případech může interpret cyklit.

10.5 Pro níže uvedenou databázi slov napište predikát `crossword/6`, který vypočítá, jak vyplnit uvedenou křížovku. První tři argumenty představují slova uváděná vertikálně zleva doprava, druhé tři argumenty představují slova uváděná horizontálně směrem shora dolů.

```
word(astante, a,s,t,a,n,t,e).
word(astoria, a,s,t,o,r,i,a).
word(baratto, b,a,r,a,t,t,o).
word(cobalto, c,o,b,a,l,t,o).
word(pistola, p,i,s,t,o,l,a).
word(statale, s,t,a,t,a,l,e).
```



Cvičení 11

11.1 V prologu implementujte vlastní verze predikátů, které pro seznamy počítají standardní seznamové funkce, které znáte z Haskellu, konkrétně imitujte funkce `last`, `init`, `zip`.

11.2 Za predpokladu existence predikátu `mf/2`, jeho prvním argumentem je číslo, vytvořte predikát, který bude imitovat chování Haskellové funkce `map`, tj. bude možné pomocí něj "aplikovat funkci `mf`"na zadaný seznam čísel.

11.3 V prologu implementujte predikát pro skalární a vektorový součin vektorů reprezentovaných jako seznam čísel.

11.4 S využitím Prologu vyřešte Einsteinovu hádanku.

Zadání

- Je 5 domů, z nichž každý má jinou barvu.
- V každém domě žije jeden člověk, který pochází z jiného státu.
- Každý člověk pije nápoj, kouří jeden druh cigaret a chová jedno zvíře.
- Žádný z nich nepije stejný nápoj, nekouří stejný druh cigaret a nechová stejné zvíře.

Nápovědy

1. Brit bydlí v červeném domě.
2. Švéd chová psa.
3. Dán pije čaj.
4. Zelený dům stojí hned nalevo od bílého.
5. Majitel zeleného domu pije kávu.
6. Ten, kdo kouří PallMall, chová ptáka.
7. Majitel žlutého domu kouří Dunhill.
8. Ten, kdo bydlí uprostřed řady domů, pije mléko.
9. Nor bydlí v prvním domě.
10. Ten, kdo kouří Blend, bydlí vedle toho, kdo chová kočku.
11. Ten, kdo chová koně, bydlí vedle toho, kdo kouří Dunhill.
12. Ten, kdo kouří BlueMaster, pije pivo.
13. Němec kouří Prince.
14. Nor bydlí vedle modrého domu.
15. Ten, kdo kouří Blend, má souseda, který pije vodu.

Úkol

- Zjistěte, kdo chová rybičky.

Postup

1. Uvažme řešení uvedené v souboru einstein_0.pl:
   `wget http://www.fi.muni.cz/ xbarnat/IB015/einstein_0.pl`
2. Pochopte, jak by měl program pracovat, a vysvětlete, proč odpověď na dotaz `?- rybicky(X).` zdánlivě cyklí.
3. Přeuspořádejte pravidla tak, aby prolog našel řešení na dotaz `?- rybicky(X).` do jedné vteřiny.
4. Odstraňte kategorizaci objektů (`barva/1`, `narod/1`, `zver/1`, `piti/1`, `kouri/1`) a požadavky na různost entit v každé kategorii. Diskutujte, v jaké situaci, by vám tato kategorizace byla prospěšná.
5. Nyní modifikujte soubor tak, aby místo predikátu `reseni/25` se použil prediát `reseni/5` (tj. sdružte entity stejného typu do seznamů).
6. Podobnou modifikaci proveďte i s jednoltivými pravidly, tj. místo `ruleX/10` použijte `ruleX/2`, a místo `ruleX/5` použijte `ruleX/1`.
7. Definujte pomocné predikáty `isLeftTo/4`,`isNextTo/4`, `isTogetherWith/4`, které prověřují požadované vztahy, například: `isNextTo(A,B,Acka,Bcka)` je pravdivý, pokud se objekt `A` vyskytuje v seznamu `Acka` na pozici, která sousedí s pozicí objektu `B` v seznamu `Bcka`. Pomocí těchto pravidel přepište všechna pravidla.
8. Obdivujte krásu výsledku po vašich úpravách.

Jiná kódování

- Reformulujte program tak, aby predikát `reseni` jako své argumenty bral seznamy objektů odpovídající jednotlivým pozicím v ulici, tj. 5 seznamů takových, že každý seznam bude obsahovat informaci o barvě domu, národnosti obyvatele, chovaném zvířeti, a oblíbeném kuřivu a nápoji obyvatele.

Cvičení 12

12.1 Určete, co počítá (jaký význam má) následující program v SWI-Prologu.

```
oOA([],[]).
oOA([H|T],[H|S]) :- delete(T,H,X), oOA(X,S).
```

12.2 Definujte predikát `nd35/1`, který je pravdivý, pokud jako parametr dostane číslo, které není bezezbytku dělitelné čísly 3 a 5.

12.3 Jaký je rozdíl mezi následujícími definicemi predikátů `member/2`. Ve kterých odpovědích se budou lišit?

a)
```
mem1(H,[H|_]).
mem1(H,[_|T]) :- mem1(H,T).
```

b)
```
mem2(H,[H|_]) :- !.
mem2(H,[_|T]) :- mem2(H,T).
```

c)
```
mem3(H,[K|_]) :- H==K.
mem3(H,[K|T]) :- H\==K, mem3(H,T).
```

12.4 S využitím knihovních funkcí pro seznamy napište predikát

a) `variace3(+List,?Y)`, kterým budete schopni generovat všechny 3-prvkové variace prvků ze seznamu `List`.

b) `kombinace3(+List,?Y)`, kterým budete schopni generovat všechny 3-prvkové kombinace prvků ze seznamu `List`, (trojice prvků ze seznamu `Y` budou uspořádány).

12.5 Predikáty z předchozího příkladu přeprogramujte tak, aby každé řešení bylo generováno pouze jednou.

12.6 Určete maximální možný výstup interpretru pro následující programy v Prologu na dotaz `?- b(X,Y).` a **zakreslete odpovídající výpočetní stromy**.

a)
```
a(X) :- X = 0.
a(X) :- X = 1,!.
a(X) :- X = 2.

b(X,Y) :- a(X),a(Y).
```

b)
```
a(X) :- X = 0.
a(X) :- X = 1.
a(X) :- X = 2.

b(X,Y) :- a(X),!,a(Y).
```

12.7 V Prologu naprogramujte predikát `prvocislo(+Num)`, který se vyhodnotí na pravda, pokud `Num` je prvočíslo. K řešení použijte naivní metodu testování zbytku po dělení všemi čísly od 2 po zadané číslo `Num`.

12.8 Napište predikát `contains(+FileName,+Char)` který se vyhodnotí na pravda, pokud v souboru se jménem `FileName`, se vyskytuje znak `Char`.

12.9 S použitím predikátů pro získání všech řešení a predikátu `contains/2` z předchozího příkladu získejte všechny znaky uvedené v daném souboru. Vyvětlete v čem je toto řešení neefektivní a navrhněte/zrealizujte řešení lepší.

Cvičení 13

13.1 S využitím mechanismu gramatik definitních klauzulí implementujte predikát `trim(+List,?TrimedList)`, který se vyhodnotí na pravda, pokud `TrimedList` vznikne odstraněním krajních (počátečních a koncových) znaků představujících prázdné místo (mezera, tabulátor a nový řádek).

13.2 S využitím mechanismu gramatik definitních klauzulí zjistěte parsováním souboru `/proc/cpuinfo` frekvenci procesoru počítače, na kterém pracujete. (Vyžaduje operační systém Linux).

13.3 Stáhněte si program sudoku.pl:
`wget http://www.fi.muni.cz/ xbarnat/IB015/sudoku.pl`.

a) Program spusťte a naučte se jej ovládat. Zamyslete se, jak byste funkcionalitu programu sami implementovali.

b) Prohlédněte si zdrojový kód programu a pochopte, jak funguje.

c) Modifikujte program tak, aby nalezená řešení splňovala podmínku, že na všech políčkách hlavní diagonály se vyskytuje pouze jedna hodnota.

13.4 S využitím knihovny `clpfd` implementujte program, který spočítá, kolika a jakými mincemi vámi používané měny lze vyskládat zadanou částku. Snažte se, aby řešení s menším počtem mincí byla preferována (nalezena dřív).

13.5 Navrhněte strukturu Prologovského programu a naprogramujte odpovídající predikáty, pomocí kterých vykreslíte na obrazovku přání do nového roku (PF-ko), které v ASCII artu vykreslí vícepatrový vánoční stromeček. Parametrizujte počet pater, jejich tvar a podobně.