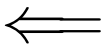
Písomná skúška FP: 30. mája 2022

Táto písomná skúška trvá 90 minút. Počas skúšky je povolené používať laptop s inštalovaným kompilátorom jazyka Haskell ghc, resp. ghci, cabal a/alebo stack, Visual Studio Code a/alebo emacs a/alebo IntelliJ IDEA. Iné používateľské aplikácie, najmä komunikačné aplikácie, webové prehliadače a aplikácie pre synchronizáciu súborov, nie sú povolené. Knihy, dokumentácia v papierovej podobe, písané poznámky atď. nie sú povolené.

Túto písomnú skúšku je potrebné vypracovať samostatne, teda bez pomoci niekoho iného a bez komunikácie s niekým iným. Odhalené podvádzanie pri skúške, napr. kopírovanie odpovedí a riešení (aj ich častí), snaha odovzdať cudzie odpovede a riešenia a pod., môže byť penalizované vylúčením študenta z predmetu, nepridelením a/alebo zrušením bodov, a to aj bodov z cvičení, prípadne aj disciplinárnym konaním v zmysle Študijného poriadku, ktoré môže viesť k vylúčeniu zo štúdia.

Meno a priezvisko: 

Študentské ID:

---

1. Prepíšte kód na 1 riadok bez použitia `do` a `<-`. (4 b.)

```
1  foo = do
2    str <- readFile "foo"
3    writeFile "bar" str
```

Riešenie:

---

2. Naimplementujte funkciu `bar`, ktorá overí, či sa prvok `x` typu `a` nachádza v poli `[a]`, a vráti `True` alebo `False`. Pokryte všetky možnosti vstupných parametrov. Prečo je potrebná typeclass Eq? (4 b.)

Riešenie:

```
1  bar :: (Eq a) => a -> [a] -> Bool
```

---

3. Prečo kód neskompiluje? Čo je potrebné zmeniť pre správnu implementáciu delenia? Prečo je táto zmena potrebná? (4 b.)

```
1  bar :: (Num a) => a -> a -> a
2  bar x y = x / y
```

Riešenie:

4. Prečo po zavolaní funkcie `foo` nastane vyhodnotenie na hodnotu `3`? Na akú hodnotu sa vyhodnotí vstupný parameter `x` funkcie `three`? (4 b.)

```
infinity :: Integer
infinity = 1 + infinity

three :: Integer -> Integer
three x = 3

foo = three infinity
```

Riešenie:

---

5. Prečo po zavolaní funkcie `foo` nenastane vyhodnotenie? Čo je potrebné zmeniť, aby vyhodnotenie nastalo v prípade, že vstup je nekonečný zoznam `[False,False,..]`? (4 b.)

```
(&&) :: Bool -> Bool -> Bool
True && x = x
False && _ = False

foo :: Bool
foo = foldl (Main.&&) False (repeat False)
```

Riešenie:

---

6. Prečo v prvom prípade nastane vyhodnotenie na `[]`, a v druhom prípade nastane chyba? (4 b.)

```
zip [] undefined  — []
zip undefined []  — *** Exception: Prelude.undefined
```

Riešenie:

7. Naimplementujte funkciu foo, ktorá pre vstup `Just [Just 1, Just 2]` vráti `Just (Just 3)`, teda spočíta `Just 1` a `Just 2`, a aplikuje `Maybe` na výsledok sčítania. (4 b.)

Riešenie:

```
1 foo x =
```

---

8. Prečo volanie `foo undefined` vráti chybu, a volanie `bar undefined` vráti `True`? Prečo používame Bang-Patterns? (4 b.)

```
1 {-# LANGUAGE BangPatterns #-}
2 foo :: p -> Bool
3 foo !x = True
4
5 bar :: p -> Bool
6 bar x = True
```

Riešenie:

9. Naimplementujte funkciu `foo`, ktorá pre dva zoznamy so vzostupnými hodnotami overí, či tieto majú rozdielne prvky. Napr. pre `[1,2,3]` a `[4,5,6]` vráti `True`, a pre `[1,2,3]` a `[3,4,5,6]` vráti `False`. Funkciu `foo` implementujte rekurzívne. Pokryte aj možnosť prázdneho zoznamu na vstupe. (4 b.)

Riešenie:

```
1 foo :: [Int] -> [Int] -> Bool
```

10. Naimplementujte funkciu `avg`, ktorá pre zoznam s číselnými hodnotami vypočíta aritmetický priemer týchto hodnôt, pričom vstupný zoznam je iterovaný len raz. Použite funkciu `uncurry` (z Prelude). (4 b.)

Riešenie:

---

11. Pomenujte a vysvetlite všetky rozdiely medzi implementáciami pomocou funkcie `foo` a `bar`. Čo majú tieto implementácie spoločné? Aký nedostatok majú obe implementácie? (5 b.)

```
1  foo :: (Integral a) => a -> a
2  foo 0 = 1
3  foo n = n * foo (n - 1)
```

```
1  bar :: (Integral a) => a -> a
2  bar n = bar' n 1
3    where
4      bar' 0 y = y
5      bar' x y = bar' (x - 1) (x * y)
```

Riešenie:

---

12. Pomenujte a vysvetlite všetky rozdiely medzi implementáciami pomocou funkcie `foo` a `bar`. Čo majú tieto implementácie spoločné? Aký nedostatok majú obe implementácie? (5 b.)

```
1  foo :: (Ord a) => [a] -> [a]
2  foo (x : xs) = foo' ++ x : foo''
3    where
4      foo' = foo [y | y <- xs, y < x]
5      foo'' = foo [y | y <- xs, y >= x]
6  foo _ = []
```

```
1  bar :: (Ord a) => [a] -> [a]
2  bar (x : xs) = bar' par (bar'' pseq (bar'' ++ x : bar'))
3    where
4      bar' = bar [y | y <- xs, y < x]
5      bar'' = bar [y | y <- xs, y >= x]
6  bar _ = []
```

Riešenie: