IB015 Neimperativní programování

Jiří Barnat

Organizace kurzu

Organizace výuky – Distanční režim

Kurz IB015

- Zakončen zkouškou.
- 6 kreditů (2/1/1+zk) = 180 hodin = 22 pracovních dnů

Přednáška

- Přednáška ve formě výukových videí (offline).

Cvičení

- Se 14 denní periodou v určené časové sloty.
- Offline videa s demonstrací řešení vybraných příkladů.
- Individuální konzultace po chatu, případně video hovoru.

Samostatné domácí úlohy

- Zadávání v interaktivní osnově předmětu.

Požadavky na ukončení

Závěrečná písemná zkouška - prezenčně

- Povinná část (Pass/Fail).
- Nepovinná část, možno získat 10 bodů.

Domácí úlohy

- Průběžné odpovědníky v ISu, časové omezení na vypracování.
- Minimálně tři větší programovací úlohy.
- Celkem možno získat 15 bodů.

Požadavky na úspěšné ukončení

- Povinná část písemky, minimálně 8 bodů z DÚ.
- Podle počtu bodů za DÚ a zkoušku:

  E:[10,12) D:[12,14) C:[14,17) B:[17,20) A:[20,25]
- Při získání 12+ bodů z DÚ je možno při úspěšném ukončení předmětu odmazat jedno hodnocení F.

Cíle kurzu

Cíle kurzu

- Studenti se seznámí s funkcionálním a logickým paradigmatem programování, díky čemuž se odprostí od imperativního způsobu uvažování o problémech a jejich řešení.
- V rámci kurzu se studenti blíže seznámí s funkcionálním programovacím jazykem Haskell a s logickým programovacím systémem Prolog.

Schopnosti absolventa

- Je schopen dekomponovat výpočetní problém na jednotlivé funkce a tuto schopnost používá při vytváření vlastních kódů i v imperativních programovacích jazycích.
- Umí efektivně použít prvky funkcionálního programování v imperativních programovacích jazycích.
- Má základní znalost programovacích jazyků Haskell a Prolog.
- Rozumí způsobu popisu programů ve funkcionálním a logickém výpočetním paradigmatu.
- Umí oddělit CO od JAK.

Požadavky a předpoklady pro úspěšné absolvování

Předpoklady

- Možné úspěšně absolvovat bez znalosti programování.
- Schopnost abstraktního myšlení.
- Základní počítačová gramotnost (Unix/Linux OS).

Znalost imperativního programování

- Je výhodou pro pochopení rozdílného způsobu myšlení v imperativním a neimperativním světě.
- Může být zpočátku mentální bariérou.

Zdroje a učební materiály

Funkcionální paradigma

- `http://haskell.cz/`
- Thompson, Simon. Haskell: the craft of functional programming.
- Structure and Interpretation of Computer Programs [`http://mitpress.mit.edu/sicp/full-text/book/book.html`]

Logické paradigma

- `http://www.learnprolognow.org`
- Nerode, Shore: Logic for Applications

Co znamená programovat?

Programování a programovací jazyk

Programování

- Vytvoření a zápis postupu řešení problému s takovou úrovní detailů a přesnosti, aby tento popis mohl být mechanicky vykonáván strojem, zejména počítačem.
- Zápis postupu = zdrojový kód programu.
- Zdrojový kód programu je uložen v textovém souboru.

Programovací jazyk

- Uměle vytvořený jazyk pro přesný a jednoznačný zápis programů člověkem.

Co znamená umět programovat?

Schopnost programovat

- **Mentální schopnost nacházet mechanicky proveditelné postupy za účelem řešení daného problému.**
- Schopnost přesně formulovat postupy v daném programovacím jazyce.

Volba a znalost programovacího jazyka

- Programovacích jazyků je mnoho.
- Volba programovacího jazyka klade omezení na způsob formulace zamýšlených postupů.

Míra zkušeností a praxe

Riziko a klam moderní doby

- Dokumentace k programovacím jazykům jsou snadno dostupné i ve formě tutoriálů, avšak samotné poznání syntaxe a sémantiky programovacího jazyka nedělá dokonalého programátora.

Míra zkušeností a praxe

Riziko a klam moderní doby

- Dokumentace k programovacím jazykům jsou snadno dostupné i ve formě tutoriálů, avšak samotné poznání syntaxe a sémantiky programovacího jazyka nedělá dokonalého programátora.

Nedokonalé vs. dokonalé

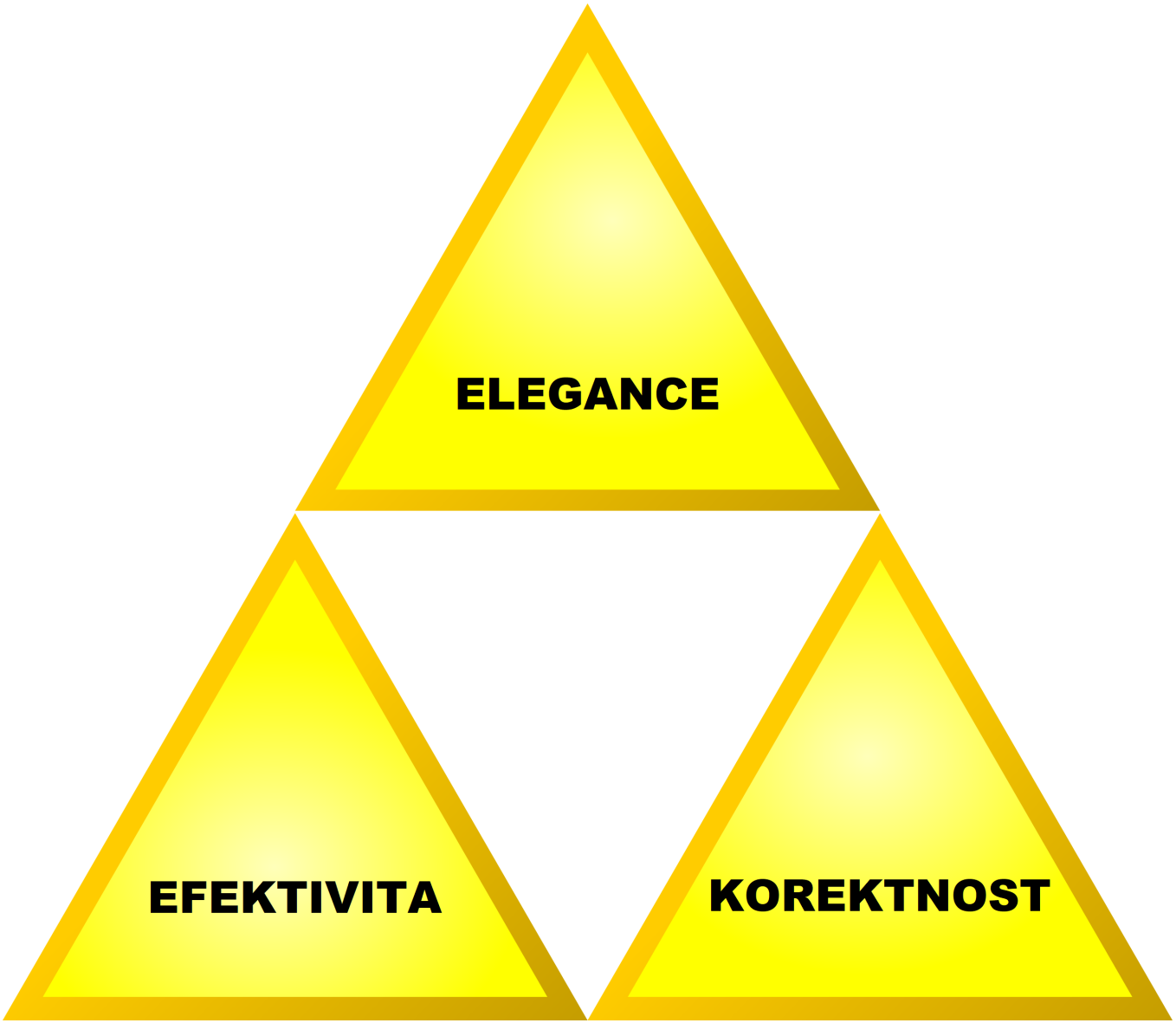
Míra zkušeností a praxe

Riziko a klam moderní doby

- Dokumentace k programovacím jazykům jsou snadno dostupné i ve formě tutoriálů, avšak samotné poznání syntaxe a sémantiky programovacího jazyka nedělá dokonalého programátora.

Nedokonalé vs. dokonalé

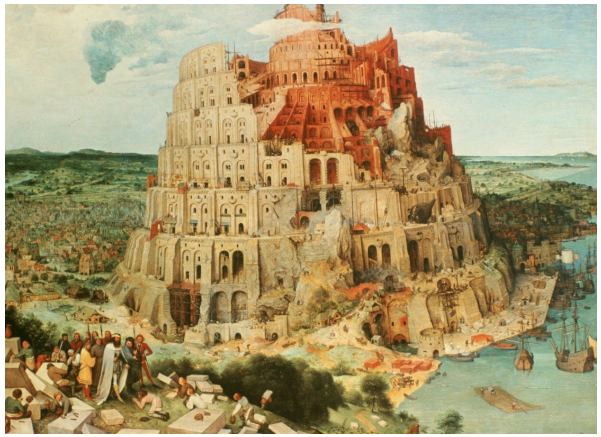
Triforce programování



Programovací jazyky

Klasifikace

- Imperativní — C/C++, Java, Python, ...
- Funkcionální – **Haskell**, OCaML, ...
- Logické – **Prolog**, ...



Jakým jazykem mluví počítač?

- Strojový kód. Program ve strojovém kódu je posloupnost čísel.
- Pro spuštění programu je potřeba provést překlad zdrojového kódu programu do strojového kódu procesoru.
- Překlad se realizuje pomocí **překladače** nebo **interpretu**.
- Pro každý programovací jazyk je potřeba jiný překladač/interpret.

Překladače a interpretry

Překladač

- Pro soubor se zdrojovým kódem programu vytvoří soubor obsahující popis programu ve strojovém kódu.
- Výsledný soubor je spustitelný.
- Pracuje se soubory.

Interpret

- Pro daný výraz / příkaz vytvoří odpovídající překlad do strojového kódu a ihned jej provede.
- Nevytváří výsledný spustitelný soubor.
- Často má možnost pracovat interaktivně.
- Pracuje s jednotlivými příkazy/výrazy.

Příklad překladu a interpretace

Programovací jazyk Haskell

- Překladač – `ghc`.
- Interaktivní interpretr – `ghci`.
- Neinteraktivní interpretace – runghc.

Překladače programovacího jazyka C/C++

- GNU C++ Compiler (g++, gcc)
- Intel C++ Compiler
- Microsoft Visual C++ Compiler

Programujeme pomocí funkcí

Co je to funkce?

Funkce v programování

- Funkce je předpis jak z nějakého vstupu vytvořit výstup.
- Transformace vstupů na výstupy musí být jednoznačná.

Příklady funkcí

- `f x = x*(x+2)`
- `objemkvadru a b c = a*b*c`
- ...

S čím funkce pracují

Typ funkce

- Vymezení objektů, se kterými daná funkce pracuje a které vrací na výstup, je součástí definice funkce. Mluvíme o tzv. **typu funkce**.

Příklady

- Funkce, která otočí obrázek o 90 stupňů směrem vpravo.

  `rotate90r :: Obrázek -> Obrázek`

- Objem kvádru.

  `objemkvadru :: Číslo × Číslo × Číslo -> Číslo`

- Počet hran polygonu.

  `hranypolygonu :: Polygon -> Celé_číslo`

Aplikace funkce – příklad

Předpoklady

```
rotate90r :: Obrázek -> Obrázek
hranypolygonu :: Obrázek -> Celé_číslo
```

 `:: Obrázek`

Aplikace funkcí

`rotate90r`  $\rightsquigarrow$ 

`hranypolygonu`  $\rightsquigarrow$ `3`

Funkce jako základní stavební kameny

Pozorování

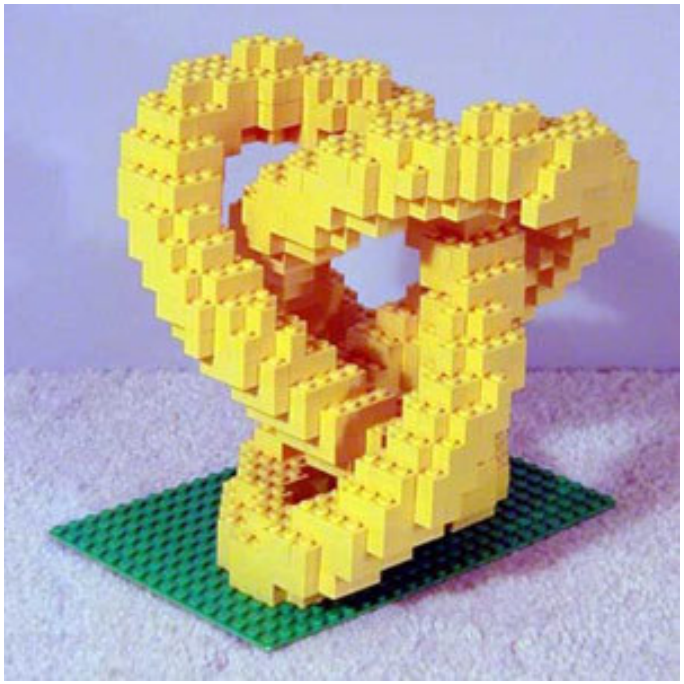
- Složitější úkony lze realizovat pomocí jednodušších operací.
- Složitější funkce lze definovat **složením** jednodušších.

Funkce jako základní stavební kameny

Pozorování

- Složitější úkony lze realizovat pomocí jednodušších operací.
- Složitější funkce lze definovat **složením** jednodušších.

Skládání – cesta ke složitějším objektům a funkcím



Skládání funkcí

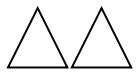
Operátor .

- `(f1 . f2) x = f1 ( f2 x )`
- Čteme jako „f1 po f2".

Příklad

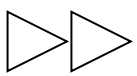
- Předpokládejme funkci `double`, která vezme obrázek a vytvoří nový zkopírováním vloženého obrázku dvakrát vedle sebe.

```
double :: Obrázek -> Obrázek
```

`double`  ⇝ 

- Novou funkci `rotate_and_double` můžeme definovat takto:

```
rotate_and_double :: Obrázek -> Obrázek
rotate_and_double x = (double . rotate90r) x
```

`rotate_and_double`  ⇝ 

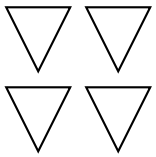
Příklady

`(rotate_and_double . rotate_and_double)`  $\rightsquigarrow$

`((double . double) . double)`  $\rightsquigarrow$

`(double . hranypolygonu)`  $\rightsquigarrow$

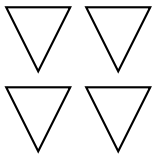
Příklady

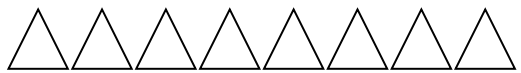
`(rotate_and_double . rotate_and_double)`  $\rightsquigarrow$ 

`((double . double) . double)`  $\rightsquigarrow$

`(double . hranypolygonu)`  $\rightsquigarrow$

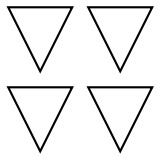
Příklady

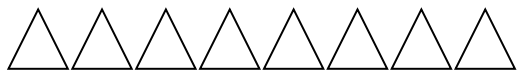
`(rotate_and_double . rotate_and_double)`  $\rightsquigarrow$ 

`((double . double) . double)`  $\rightsquigarrow$ 

`(double . hranypolygonu)`  $\rightsquigarrow$

Příklady

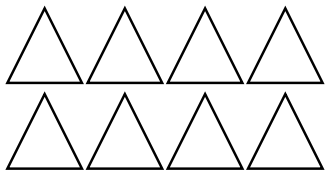
`(rotate_and_double . rotate_and_double)` △ ⇝ 

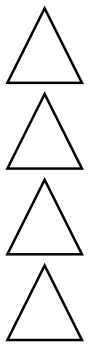
`((double . double) . double)` △ ⇝ 

`(double . hranypolygonu)` △ ⇝ **ERROR**

Příklady – pokračování

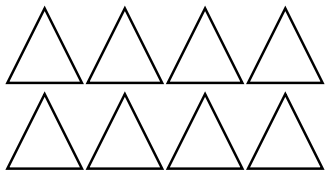
Jak pomocí `double`, `rotate90r` a  vyrobit následující?

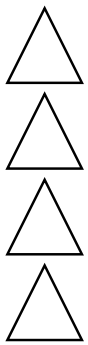
a) 

b) 

Příklady – pokračování

Jak pomocí `double`, `rotate90r` a  vyrobit následující?

a) 

b) 

Řešení

a) (r.r.r.d.r.d.d) 

b) (r.r.r.d.d.r) 

Skládání funkcí a priority operací v Haskellu

Složené funkce a η-redukce

- Složení funkcí je možné definovat bez uvedení parametru.
- Tj. definici

```
rotate_and_double x = (double.rotate90r) x
```

  lze zapsat také jako

```
rotate_and_double = double.rotate90r
```

POZOR na prioritu vyhodnocování v Haskellu

- Aplikace funkce na parametry má nejvyšší prioritu.

`double.rotate90r` △ = `double.(rotate90r` △`)` ⇝ **ERROR**

- Závorky kolem výrazu `double.rotate90r` jsou při aplikaci na hodnotu △ nutné.

Terminologie

Typová signatura:

```
rotate_and_double :: Obrázek ->Obrázek
```

Jméno funkce

```
rotate_and_double x = (double.rotate) x
```

Tělo funkce

```
rotate_and_double x = (double.rotate) x
```

Definice funkce

```
rotate_and_double x = (double.rotate) x
```

Terminologie – pokračování

Formální parametr

```
rotate_and_double x = (double.rotate) x
```

Aktuální parametr

```
rotate_and_double
```

Výraz

rotate_and_double

Podvýraz

```
rotate_and_double (
```
rotate_and_double

```
)
```

Funkcionální programování v Haskellu

Funkcionální výpočetní paradigma

- program = výraz + definice funkcí
- výpočet = úprava (zjednodušování) výrazu
- výsledek = hodnota (nezjednodušitelný tvar výrazu)

Příklad programu

- definice funkcí

```
square x = x * x
pyth a b = square a + square b
```

- výraz

```
pyth 3 4
```

Výpočet funkcionálního programu

Program

- definice funkcí

```
square x = x * x
pyth a b = square a + square b
```

- výraz

```
pyth 3 4
```

Výpočet

```
pyth 3 4 ⇝ square 3 + square 4 ⇝ 3 * 3 + square 4 ⇝
         ⇝ 3 * 3 + 4 * 4 ⇝ 9 + 4 * 4 ⇝ 9 + 16 ⇝
         ⇝ 25
```

Lokální definice

- Definují symboly (funkce, konstanty) pro použití v jednom výrazu, vně tohoto výrazu jsou tyto symboly nedefinované.
- Lokální definice mají vyšší prioritu než globální definice.

V Haskellu pomocí let ... in

- **let** definice **in** výraz

```
let fcube x = x * x * x in fcube 12

let fcube x = x * x * x in let c = 12 in fcube c

let fcube x = x * x * x; c = 12 in fcube c
```

Základy Haskellu – Základní datové typy

Čísla

- `Integer` – libovolně velká celá čísla
- `Int` – celá čísla do velikosti slova procesoru
- `Float` – reálná čísla
- `Rational` – racionální čísla

Znaky a řetězce

- `Char` – znak, příklady hodnot: `'a'`, `'2'`, `'>'`
- `String` – řetězec, například: `"Toto je řetězec."`
- `String` je totéž co `[Char]`

Pravdivostní hodnoty

- `Bool`
- Typ `Bool` má pouze 2 hodnoty: `True` a `False`

Základy Haskellu – Víceřádkové definice

Příklad

- Definujte funkci `jedna_nebo_dva`, která vrací `True` pokud dostane na vstupu číslo `1` nebo `2`, jinak vrací `False`.

```
jedna_nebo_dva :: Integer -> Bool
jedna_nebo_dva 1 = True
jedna_nebo_dva 2 = True
jedna_nebo_dva _ = False
```

Víceřádkové definice funkcí

- Na místě formálních parametrů se použijí tzv. vzory.
- Použije se první vzor, který vyhovuje, nic jiného.
- Symbol _ vyhovuje libovolnému parametru.
- Lze použít pro větvení výpočtu.

Základy Haskellu – Větvení výpočtu

Podmíněný výraz

- **if** *podmínka* **then** *výraz1* **else** *výraz2*
- *podmínka* – výraz, který se vyhodnotí na hodnotu typu `Bool`
- *výraz1* se vyhodnotí pokud se podmínka vyhodnotí na hodnotu `True`, *výraz2* se vyhodnotí, pokud se podmínka vyhodnotí na hodnotu `False`.
- Výrazy *výraz1* a *výraz2* musejí být stejného typu.

Test na rovnost

- Pro dotaz na rovnost používáme symbol `==`.
- `3 == 4` $\rightsquigarrow$ `False`
- `3 = 4` $\rightsquigarrow$ **Error**

Základy Haskelu – Infix, Prefix, Parametry

Možnosti zápisu binárních funkcí

- Infixový zápis binárních funkcí: `3+4`, `4*5`
- Prefixový zápis binárních funkcí: `(+) 3 4`, `(*) 4 5`

Volání funkce a parametry

- Jméno funkce a použité parametry jsou odděleny mezerou, pokud je některý z parametrů výraz, který je sám o sobě aplikace funkce na argumenty, je třeba celý tento výraz ozávorkovat.
- `(*) 3 4 + 5` $\rightsquigarrow$ `17`
- `(*) 3 + 4 5` $\rightsquigarrow$ **Error**
- `(*) 3 (+) 4 5` $\rightsquigarrow$ **Error**
- `(*) 3 ( (+) 4 5 )` $\rightsquigarrow$ `27`

První spustitelný program

Zde byla ukázka jednoduchého programu v Haskellu, který bylo možné přeložit do spustitelného programu a spustit.

Na základě zkušeností a četných žádostí cvičících byl tento příklad z první přednášky odstraněn.

Checkpoint

Co to je?

- Úkol, který je možné vyřešit na základě faktů doposud uvedených na přednáškách.
- Slouží ke kontrole, že zvládám to, co bych už měl(a) umět.

Checkpoint

- V programovacím jazyce Haskell napište funkci, která bude řešit dělitelnost dvou celočíselných čísel, tj. pro své dva celočíselné argumenty dělence a dělitele, rozhodne, zda je zadaný dělenec dělitelný beze zbytku zadaným dělitelem a to tak, že nepoužije operace pro dělení `/` ani počítání zbytku po dělení `mod`, je povoleno použít operaci celočíselného dělení `div`.
- Funkci otestujte s použitím interpretu jazyka Haskell.