

# Paradigmata programování

**Karel Břinda**

brinda@jfji.cvut.cz

FJFI ČVUT v Praze

25. listopadu 2010

# Co to je paradigma

## Slovo „paradigma“

- Z řeckého *parádeigma* = vzor, příklad, model
- Vzorec myšlení, vzor vztahů
- Ve vědě - souhrn základních domněnek, předpokladů, představ dané skupiny vědců

**Paradigma programování** - styl, jakým lze programovat v daném jazyku (pokud ho podporuje)

# Paradigmata programování

## Základní rozdělení

- **Imperativní (Procedurální) programování** - „řekneme, jak se to dělá“
  - **Nestrukturované programování** - používání návěští, např. původní *BASIC*, *COBOL* či *FORTRAN*
  - **Strukturované programování** - podmíněné cykly
- **Deklarativní programování** - „řekneme, co se dělá“
  - **Funkcionální programování** - výpočet sestává ze zřetězování funkcí, např. *HASKELL*
  - **Logické programování** - založeno na matematické logice - např. *PROLOG*
  - **Programování s omezujícími podmínkami**
- **Objektově orientované programování** - vzájemné interakce objektů, např. *SMALLTALK*
- **Paralelní programování** - důraz na souběžné vykonávání, např. *MPD*

# Reálně

- Žádné paradigma se **nepoužívá v čisté podobě**
- Většina dnešních jazyků - **multiparadigmatické** (*C++*, *PYTHON*, *F#* ...)
- Kombinace např.:
  - GUI - objektově orientované
  - výpočty - procedurální či funkcionální

# Stejný kód procedurálně a funkcionálně

## Listing 1: Procedurální přístup v jazyku PYTHON

```
target = []  
for item in source_list:  
    trans1 = G(item)  
    trans2 = F(trans1)  
    target.append(trans2)
```

## Listing 2: Funkcionální přístup v jazyku PYTHON

```
compose2 = lambda F, G: lambda x: (F(G(x)))  
target = map(compose2(F,G), source_list)
```

# Imperativní (procedurální) programování

- Popis programu krok po kroku (imperativ = rozkaz)
- Procedurální - používání podprogramů

# Nestrukturované programování

- První paradigma pro jazyky **Turing-kompletní** (= ekvivalentní **Turingovu stroji**)
  - Stejná výpočetní síla jako Turingův stroj
  - Více dle *Churchovy-Turingovy* teze konečným zařízením spočítat nelze
  - Důsledek: úlohu, kterou neumíme vyřešit v takovémhle jazyce, nevyřešíme v žádném
- Lineární sekvence příkazů, skoky příkazem **GOTO** na číslo řádku (případně na návěští) - „špagety styl“
- Možnost skákat i dovnitř do podprogramů
- Rekurze obtížná - neexistuje mechanismus předávání proměnných funkci přes zásobník (tak, jak ho známe dnes)
- Pouze základní datové typy (čísla, řetězce)

# Nestrukturované programování

## Listing 3: Příklad nestrukturovaného programování v jazyku BASIC

```
1 INPUT "What is your name: ", U$
2 PRINT "Hello "; U$
3 INPUT "How many stars do you want: ", N
4 S$ = ""
5 FOR I = 1 TO N
6 S$ = S$ + "*"
7 NEXT I
8 PRINT S$
9 INPUT "Do you want more stars? ", A$
0 IF LEN(A$) = 0 THEN GOTO 9
1 A$ = LEFT$(A$, 1)
2 IF A$ = "Y" OR A$ = "y" THEN GOTO 3
3 PRINT "Goodbye "; U$
4 END
```

# Strukturované programování

- „Pokročilejší“ podmínky, vícecestné větvení
- „Pokročilejší“ cykly
- Zákázáno používání příkazu **GOTO**
- Kromě základních datových typů už i složitější (pole, záznam. . . )
- Výjimky

# Deklarativní programování

- Program popisuje, **co se má vypočítat**, nikoliv jak

# Funkcionální programování

## $\lambda$ -kalkul - matematický základ funkcionálního programování

- 30. léta, Alonzo Church
- Studuje funkce (funkce = metoda výpočtu) a rekurze
- Ekvivalentní konceptu **Turingova stroje**
- Byl pomocí něj poprvé rozřešen **Entscheidungsproblem** (problém rozhodnutelnosti)
  - „Existuje algoritmus, který umí rozhodnout, zda je dané matematické tvrzení v daném formálním jazyce pravdivé nebo nepravdivé?“

# Funkcionální programování

- Dekompozice řešeného problému do množiny funkcí bez vedlejšího efektu (pro daný vstup vrací výstup a nedělají nic jiného)
- Program - deterministické vyhodnocování matematických výrazů (analogií *Microsoft Excel*)
- Jazyky neobsahují přiřazení a cykly (místo nich se používá rekurze), apod...
  - **Není možnost přiřazení do již existující proměnné!**

# Funkcionální programování

## Listing 4: Faktoriál v jazyku HASKELL

```
faktorial 0 = 1
faktorial n = n * faktorial (n - 1)
```

## Listing 5: Quick sort v jazyku HASKELL

```
qsort [] = []
qsort (x:xs) = qsort mensi ++ [x] ++ vesti_rovno
  where
    mensi = [ y | y <- xs, y < x ]
    vetsi_rovno = [ y | y <- xs, y >= x ]
```

# Koncepty funkcionálního programování

- **Funkce první třídy a funkce vyšších řádů**
- **Čisté funkce**
- **Rekurze**
- **Striktní, nestriktní a líné vyhodnocení**

# Koncepty funkcionálního programování

## Funkce první třídy

Vlastnost jazyka; znamená, že funkce může vzniknout za běhu programu, být uložena v proměnné a předána jako argument jiné funkci

## Funkce vyššího řádu

Funkce, které mohou převzít jako argument jiné funkce a mohou vrátit funkci

## Example (Funkce vyššího řádu)

Derivace, neurčitý integrál funkce, součin funkcí. . .

## Example (Funkce vyššího řádu)

Funkce `map(funkce, seznam)`

# Koncepty funkcionálního programování

## Čisté funkce

Funkce, které nemají žádné vedlejší účinky (paměťové, I/O - např. nemění globální proměnné). Výhody:

- 1 Pokud se výstup funkce na nic nepoužívá, nemusí se funkce volat
- 2 Čisté argumenty  $\Rightarrow$  funkce lze vyhodnocovat v libovolném pořadí, dokonce paralelně
- 3 Eliminace společného podvýrazu - dva výrazy mají stejné hodnoty  $\Rightarrow$  jeden lze dosadit za druhý

# Koncepty funkcionálního programování

## Rekurze

- **Vnořená rekurze** - rekurzivní volání v argumentech funkce, např. *Ackermannova funkce*
- **Stromová rekurze** - ve stejném výrazu několik rekurzivních volání (bez vzájemného vnoření)
- **Lineární rekurze** - v každé z alternativ v definici funkce max. jedno rekurzivní volání
- **Koncová rekurze** - případ lineární rekurze, rekurzivní volání poslední operací každé alternativy v definici funkce

# Koncepty funkcionálního programování

## Listing 6: Demonstrace různých přístupů k vyhodnocování výrazů

```
f(x) := x^2+x+1  
g(y, z) := y+z  
f(g(1, 4))
```

### Striktní vyhodnocení

- Argumenty funkce vyhodnoceny před jejím voláním
- Používá se např. v jazyku *LISP*
- Výpočet se provádí jednou, efektivní

```
f(g(1, 4))  ->  f(1+4)  ->  
-> f(5)     ->  5^2+5+1  ->  31
```

# Koncepty funkcionálního programování

## Listing 6: Demonstrace různých přístupů k vyhodnocování výrazů

```
f(x) := x^2+x+1
g(y, z) := y+z
f(g(1, 4))
```

### Nestriktní vyhodnocení

- Argumenty přenechány funkci nevyhodnocené
- Umožňuje implementovat nekonečné struktury
- Méně efektivní

```
f(g(1, 4))    ->    g(1,4)^2+g(1,4)+1 ->
->    (1+4)^2+(1+4)+1    ->    5^2+5+1    ->    31
```

# Koncepty funkcionálního programování

## Listing 6: Demonstrace různých přístupů k vyhodnocování výrazů

```
f(x) := x^2+x+1  
g(y, z) := y+z  
f(g(1, 4))
```

### Líné vyhodnocení

- Druh nestriktního vyhodnocení
- Argument se vždy počítá maximálně pouze jednou
- Hlavně u čistě funkcionálních jazyků (*MIRANDA*, *HASKELL*)

# Seznamy

- Základním kamenem funkcionálních jazyků (viz např. jazyk *LISP* - akronym List Processing)
- Např. seznam čísel [1,2,3,4,5], seznam písmen (řetězec) ['l', 'a', 'm', 'b', 'd', 'a']
- 2 přístupy
  - Konstruktivní – seznamy se nemění, pouze se vytvářejí nové
  - Destruktivní – již vytvořený seznam se může změnit

# Funkcionální programování

## Výhody

- Formální dokazování správnosti počítačových programů
- Modularita
- Jednoduché ladění a testování
- Paralelizace výpočtů
- Velikost kódu (funkcionální programy ve srovnání s procedurálními 2x – 10x kratší)

## Nevýhody

- Pomalejší výpočet (ale max. logaritmické zpomalení)
- Větší náročnost na systémové prostředky

# Objektově orientované programování

- Založeno na myšlence vzájemné interakce objektů
- Komunikace objektů pomocí zasílání zpráv
- Snaha o odstranění cyklů pomocí iterátorů
- Snaha o odstranění podmínek pomocí výjimek

# Paralelní programování

- Důraz souběžné vykonávání programu
- Musí být k dispozici prostředky pro vzájemnou synchronizaci vláken, sdílení dat, atd. . .

# Reference

- J. Konečný, V. Vychodil: **Paradigmata programování 1A** (skripta), <http://phoenix.inf.upol.cz/esf/ucebni/pp1a.pdf>.
- M. Beneš: **Funkcionální programování** (web k předmětu), <http://www.cs.vsb.cz/navrat/vyuka/flp/texty/fp/index.html>.
- Wikipedia.org, hesla *Programming Paradigma*, *Declarative Programming*, *Functional Programming*, *Procedural Programming*, *Imperative Programming*, *Haskell*, *Python*.
- G. van Rossum: **Functional Programming HOWTO** (Release 3.1.2), Python documentation, <http://docs.python.org/py3k/howto/functional.html>.
- D. Leavitt: **Muž, který věděl příliš mnoho - Alan Turing a první počítač** (kniha), Argo, Dokořán, 2007.
- P. Almásy: **Haskell a funkcionální programování** (seriál na Root.cz), <http://www.root.cz/clanky/haskell-a-funkcionalni-programovani/>.
- J. Dvořák: **Funkcionální programování - Úvod do jazyka Lisp** (prezentace), <http://www.uai.fme.vutbr.cz/~jdvorak/Vyuka/Jui/Predn1.ppt>.