# Uma Breve Introdução

Felipe A. Lessa

Departamento de Ciência da Computação
Universidade de Brasília

## Assuntos de hoje



## Assuntos de hoje



## Um pouco sobre mim

- Sou Felipe Lessa, estou no 4º semestre.
- Programo com Haskell há mais ou menos um ano.

## E vocês

- Se apresentem =).

## Assuntos de hoje



## O que esperar

1. Diversão. Se estiver meio chato, é um bug. XD
2. Aprender a programar em Haskell.
3. Aprender a gostar de Haskell (esta é fácil!).
4. Aprender a *pensar funcionalmente*.

## O que **não** esperar

- Aprender programação em geral.
  - Não há tempo para isso.
  - Pulamos a salada e vamos direto para a sobremesa.
- Aprender passivamente.
  - Para aprender é necessário meter a mão na massa.
  - Vou assumir que vocês vão fazer por onde =).
- Muito tempo pro almoço.

## Assuntos de hoje



## A panacéia da computação?

"Se acabaram-se os seus problemas?"

## Características principais da linguagem

- De propósito geral.
- Funcional, $\lambda$s são cidadãos de primeira classe.
- Pura, sem efeitos colaterais.
- Estaticamente tipada com inferência de tipos.
- Fortemente tipada (i.e. com esteróides).
- Polimórfica (i.e. objetos não fazem falta).
- Com tipos algébricos de dados e funções de alta ordem.
- Monádica (mais simples que você imagina).

## Por que usar?

- Aumento em produtividade.
- Código mais claro, conciso e manutenível.
- Fácil modularização.
  (leia "Why functional programming matters?")
- Alta confiabilidade (sem core dumps, sem ponteiros nulos).
- Fácil conferência (e.g. equational reasoning, QuickCheck).

## Principais implementações

**GHC** Compilador com otimizador muito maduro e que gera código de muito boa performance *hoje*, além de existir muita pesquisa em como extrair mais do silício. Também possui interpretador, e é feito totalmente em Haskell.

**Hugs** Interpretador conhecido pelos alunos do Ladeira. Implementa todo o padrão Haskell 98 e mais algumas extensões (i.e. não é um brinquedo!), mas possui performance fraca.

**Yhc** Está ainda engatinhando porém promete competir pau-a-pau com o Glorious Glasgow Haskell Compiler.

## Assuntos de hoje



## "Hello world!"

```
1 main = putStrLn "Hello world!"
```

## "Hello world!"

- Você acabou de ver um monad e nem percebeu =).
- Veja que não declaramos tipos, porém todos foram inferidos e verificados.

## Função fatorial

### Definição da função fatorial

$$n! = \prod_{i=1}^{n} i = \begin{cases} 1, & n \leq 1 \\ n \cdot (n-1)!, & \text{caso contrário} \end{cases}$$

```
1 fatorial n = product [1..n]
```

- Tradução direta da linguagem matemática, ou seja, muito mais fácil de entender que um loop `for`.
- Note que essa função é polimórfica!
  Outro dia veremos seu tipo.

## Função fatorial II

### Definição da função fatorial

$$n! = \prod_{i=1}^{n} i = \begin{cases} 1, & n \leq 1 \\ n \cdot (n-1)!, & \text{caso contrário} \end{cases}$$

```
1 fatorial 1 = 1
2 fatorial n = n * fatorial (n - 1)
```

## Função fatorial III

### Definição da função fatorial

$$n! = \prod_{i=1}^{n} i = \begin{cases} 1, & n \leq 1 \\ n \cdot (n-1)!, & \text{caso contrário} \end{cases}$$

```
1 fatorial n = fat' 1 n
2   where
3     fat' acc 1 = acc
4     fat' acc n = fat' (acc * n) (n - 1)
```

## Função fatorial IV

### Definição da função fatorial

$$n! = \prod_{i=1}^{n} i = \begin{cases} 1, & n \leq 1 \\ n \cdot (n-1)!, & \text{caso contrário} \end{cases}$$

```
1 fatorial n = foldl (*) 1 [1..n]
```

- Parece com a primeira implementação?
- Na verdade, podemos definir `product = foldl (*) 1`.

## Ordenar linhas

```haskell
import Data.List

ordenar str =
  let  linhas   = lines str
       ordenado = sort linhas
  in   unlines ordenado

main = do
  entrada ← getContents
  putStrLn (ordenar entrada)
```

## Ordenar linhas II

```haskell
import Data.List
main = interact (unlines . sort . lines)
```

## K-way merge

```haskell
import System -- Usaremos no proximo slide

merge2 xs     []     = xs
merge2 []     ys     = ys
merge2 (x:xs) (y:ys) =
  case x `compare` y of
    EQ → x : merge2 xs ys
    LT → x : merge2 xs (y:ys)
    GT → y : merge2 (x:xs) ys

merge [x]         = x
merge []          = []
merge (x1:x2:xs) =
  let  x12 = merge2 x1 x2
  in   merge (x12 : merge xs)
```

## K-way merge (cont.)

```haskell
main = do
  args ← getArgs
  arquivos ← mapM readFile args
  let  linhas = [lines arq | arq ← arquivos]
       merged = unlines (merge linhas)
  writeFile "saida.txt" merged
```

## Assuntos de hoje



## Dever de casa! =)

- Instalar o GHC no seu computador (veja o site `www.haskell.org`).
- Começar a ler o Wikibook (`en.wikibooks.org/wiki/Haskell`).