Introdução

MCTA016-13 - Paradigmas de Programação

Emilio Francesquini
e.francesquini@ufabc.edu.br

2019.Q2

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

Disclaimer

UFABC

- Estes slides foram preparados para o curso de **Paradigmas de Programação na UFABC**.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- Conteúdo baseado no texto preparado, e gentilmente cedido, pelo Professor Fabrício Olivetti de França da UFABC.

Paradigmas de Programação

Paradigmas de Programação → Haskell 🙂

Haskell

- Surgiu em 1990 com o objetivo de ser a primeira linguagem puramente funcional.
- Por muito tempo considerada uma linguagem acadêmica.
- Atualmente é utilizada em diversas empresas (totalmente ou em parte de projetos).

Haskell: Características

UFABC

Por ter sido criada por um comitê de estudiosos de linguagem de programação funcional e com a mentalidade de mantê-la útil para o ensino e pesquisa de linguagem de programação, assim como uso em empresas, a linguagem adquiriu diversas características distintas e interessantes não observadas em outras linguagens.

UFABC

- **Códigos concisos e declarativos:** o programador *declara* o que ele quer ao invés de escrever um passo-a-passo. Programas em Haskell chegam a ser dezenas de vezes menores que em outras linguagens.

```haskell
take 100 [x | x <- N, primo x]
```

UFABC

- Sistema de tipagem forte: ao contrário de linguagens como *Java* e *C*, as declarações de tipo no Haskell são simplificadas (e muitas vezes podem ser ignoradas), porém, seu sistema rigoroso permite que muitos erros comuns sejam detectados em tempo de compilação.

```
1  int x    = 10;
2  double y = 5.1;
3  System.out.println("Resultado:  " + (x*y));
```

OK!

UFABC

- Sistema de tipagem forte: ao contrário de linguagens como *Java* e *C*, as declarações de tipo no Haskell são simplificadas (e muitas vezes podem ser ignoradas), porém, seu sistema rigoroso permite que muitos erros comuns sejam detectados em tempo de compilação.

```
1  x = 10  :: Int
2  y = 5.1 :: Double
3  print ("Resultado: " + (x*y) )
```

ERRO!

UFABC

- **Compreensão de listas:** listas são frequentemente utilizadas para a solução de diversos problemas. O Haskell utiliza listas como um de seus conceitos básicos permitindo uma notação muito parecida com a notação de conjuntos na matemática.

$$xs = \{x \mid x \in \mathbb{N}, x \text{ impar}\}$$

```haskell
xs = [x | x <- N, impar x]
```

UFABC

- Imutabilidade: não existe um conceito de variável, apenas nomes e declarações. Uma vez que um nome é declarado com um valor, ele não pode sofrer alterações.

```
1  x = 1.0
2  x = 2.0
```

ERRO!

UFABC

- **Funções Recursivas:** com a imutabilidade, o conceito de laços de repetição também não existe em linguagens funcionais. Eles são implementados através de funções recursivas.

```c
int x = 1;
for (int i = 1; i <= 10; i++) {
    x = x * 2;
}
printf("%d\n", x);
```

UFABC

- **Funções Recursivas:** com a imutabilidade, o conceito de laços de repetição também não existe em linguagens funcionais. (Por que?) Eles são implementados através de funções recursivas.

```haskell
f 0 = 1
f n = 2 * f (n - 1) -- Note que f(x) é o mesmo que f x
print (f 10)
```

UFABC

- **Funções de alta ordem:** funções podem receber funções como parâmetros. Isso permite definir funções genéricas, compor duas ou mais funções e definir linguagens de domínio específicos (ex.: *parsing*).

```
1 print (aplique dobro [1,2,3,4])
2 > [2,4,6,8]
```

UFABC

- Tipos polimórficos: permite definir funções genéricas que funcionam para classes de tipos. Por exemplo, o operador de soma `+` pode ser utilizado para qualquer tipo numérico.

```
1 1 + 2           -- 3
2 1.0 + 3.0       -- 4.0
3 (2%3) + (3%6) -- (7%6)
```

UFABC

- Avaliação preguiçosa: ao aplicar uma função, o resultado será computado apenas quando requisitado. Isso permite evitar computações desnecessárias, estimula uma programação modular e permite estruturas de dados infinitos.

```haskell
listaInf = [1..] -- 1, 2, 3, ...
print (take 10 listaInf)
```

UFABC

- Raciocínio equacional: podemos usar expressões algébricas para otimizar nosso programa ou provar sua corretude.

muito cedo para dar um exemplo...

Ambiente de Programação

GHC e GHCI

UFABC

- GHCi Haskell Compiler (GHC): compilador de código aberto para a linguagem Haskell.
  - Padrão de fato
  - Outros compiladores existem mas são incompletos ou têm uma equipe limitada de manutenção
- Possui um modo interativo ghci (similar ao iPython).
  - REPL - Read, Evaluate, Print, Loop

Uso recomendado de:

- Git - controle de revisão
- Stack - gerenciamento de projeto e dependências
- Haddock - documentação

UFABC

Haskell Platform

- `https://www.haskell.org/downloads#platform`
- Para o Linux escolha a distribuição *Generic*, mesmo que tenha pacote para sua distribuição.
- Vários sabores de SOs disponíveis

UFABC

- Haskell Stack
  - ATENÇÃO!!!! NÃO UTILIZE O APT-GET PARA INSTALAR O GHC OU O STACK!
  - Para instalar o Stack no Linux :

```
1 curl -sSL https://get.haskellstack.org/ | sh
```

ou

```
1 wget -qO- https://get.haskellstack.org/ | sh
```

  - Para instalar no Windows (você quer mesmo fazer isso?) faça o download do instalador no site https://docs.haskellstack.org/

Verificando a instalação

UFABC

```
> stack new primeiroProjeto simple
> cd primeiroProjeto
> stack setup
> stack build
> stack exec primeiroProjeto
```

Editores recomendados

UFABC

- Diversos editores de texto tem suporte à edição, compilação e execução de código Haskell. Entre eles estão Emacs, Vim, Atom, Sublime e Visual Studio Code. Todos baseados no Intero, um backend para IDEs de Haskell.
- Fique a vontade para escolher o editor da sua preferência. Em seguida descrevemos as instruções para utilizar o Intero com o Atom e com o MS Visual Studio Code.

Editores recomendados - Atom

UFABC

- Atom - `https://atom.io/`
  - Com os pacotes:
    - haskell-grammar
    - language-haskell

Editores Recomendados - VSCode

UFABC

Acesse `https://code.visualstudio.com/` e baixe a versão compatível com o seu SO.

Após o download, nas máquinas com Ubuntu do laboratório:

```
sudo dpkg -i nome_do_arquivo.deb
```

Em seguida, precisamos instalar o Intero, hlint.

```
stack install intero hlint apply-refact
```

Com a instalação concluída, abra o Visual Studio Code, no canto inferior esquerdo clique na engrenagem, Extensions e instale a extensão Haskero e haskell-linter.

UFABC

Se você tiver instalado o GGC usando Stack, substitua `ghci` abaixo por `stack ghci`

```
$ ghci
> 2+3*4
14

> (2+3)*4
20

> sqrt (3^2 + 4^2)
5.0
```

Interpretador GHCi

A função de exponenciação (^) tem prioridade maior do que multiplicação e divisão (*, /) que por sua vez tem prioridade maior que a soma e subtração (+, -).

```
$ ghci
> 2+3*4^5 == 2 + (3 * (4^5))
```

Informação de operadores

UFABC

Para saber a prioridade de um operador basta digitar:

```
> :i (+)
class Num a where
  (+) :: a -> a -> a
  ...
    -- Defined in ‘GHC.Num’
infixl 6 +
```

A informação indica que `+` é um operador que pode ser utilizado para qualquer tipo numérico, tem precedência nível 6 (quanto maior o número maior sua prioridade) e é associativo a esquerda. Ou seja: `1 + 2 + 3` vai ser computado na ordem `(1 + 2) + 3`.

Funções

- Na matemática a aplicação de funções em seus argumentos é definida pelo nome da função e os parâmetros entre parênteses.
- A expressão $f(a, b) + c * d$ representa a aplicação de f nos parâmetros a e b e, em seguida, a soma do resultado com o resultado do produto entre c e d.
- Em Haskell, a aplicação de função é definida como o nome da função seguido dos parâmetros separados por espaço com a maior prioridade na aplicação da função. O exemplo anterior ficaria:

```
1 f a b + c * d
```

UFABC

A tabela abaixo contém alguns contrastes entre a notação matemática e o Haskell:

Matemática	Haskell
$f(x)$	f x
$f(x, y)$	f x y
$f(g(x))$	f (g x)
$f(x, g(y))$	f x (g y)
$f(x)g(y)$	f x * g y

UFABC

Criem um arquivo *teste.hs*, abram no editor e no mesmo diretório iniciem o GHCi. No arquivo digitem:

```
1 dobra x = x + x
2
3 quadruplica x = dobra (dobra x)
```

UFABC

No GHCi:

```
> :l teste.hs
> quadruplica 10
40
```

O comando `:l` carrega as definições contidas em um arquivo fonte.

UFABC

Acrescentem a seguinte linha no arquivo fonte:

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

e no GHCi:

```
1 > :reload
2 > fatorial 5
3 120
```

Outros comandos do GHCi

UFABC

O comando `:t` mostra o tipo da função enquanto o comando `:q` sai do ghci.

```
> :t dobra
dobra :: Num a => a -> a

> :q
$
```

Outras coisas interessantes a saber

UFABC

- `:h` - Imprime a ajuda
- `:{` seguido de comandos e finalizado por `:}` permite comandos com múltiplas linhas
  - Também é possível separar as linhas com `;`

```
1  > :{
2  | fatorial 0 = 1
3  | fatorial n = n * fatorial (n - 1)
4  | :}
5  > fatorial 5
6  120
7  > fatorial2 0 = 1 ; fatorial2 n = n * fatorial2 (n - 1)
8  > fatorial2 7
9  5040
```

Convenções

Requisitos dos nomes

UFABC

- Os nomes das funções e seus argumentos devem começar com uma letra minúscula e seguida por 0 ou mais letras, maiúsculas ou minúsculas, dígitos, *underscore*, e aspas simples:

```
funcao, ordenaLista, soma1, x'
```

Nomes reservados

UFABC

- Os únicos nomes que não podem ser utilizados são:

```
case, class, data, default, deriving do, else,
foreign, if, import, in, infix, infixl, infixr,
instance, let module, newtype, of, then, type,
where
```

Convenção para listas

UFABC

- As listas são nomeadas acrescentando o caractere ’s’ ao nome do que ela representa.
- Uma lista de números `n` é nomeada `ns`, uma lista de variáveis `x` se torna `xs`. Uma lista de listas de caracteres tem o nome `css`.

Regra de layout

UFABC

- O layout dos códigos em Haskell é similar ao do Python, em que os blocos lógicos são definidos pela indentação.

```haskell
f x = a*x + b
      where
        a = 1
        b = 3
z = f 2 + 3
```

- A palavra-chave `where` faz parte da definição de `f`, da mesma forma, as definições de `a` e `b` fazem parte da cláusula `where`. A definição de `z` não faz parte de `f`.

Tabs vs Espaço

UFABC

- A definição de tabulação varia de editor para editor.
- Ainda que seja o mesmo editor, a tabulação varia de usuário para usuário.
- Como o espaço é importante no Haskell, usem espaços em vez de tab.
- Use Emacs. 😛

https://www.youtube.com/watch?v=SsoOG6ZeyUI

UFABC

Comentários em uma linha são demarcados pela sequência --, comentários em múltiplas linhas são demarcados por {- e -}:

```
1  -- função que dobra o valor de x
2  dobra x = x + x
3
4  {-
5  dobra recebe uma variável numérica
6  e retorna seu valor em dobro.
7  -}
```

Primeiro Projeto

Primeiro projeto compilável

UFABC

- Para criar projetos, utilizaremos a ferramenta stack. Essa ferramenta cria um ambiente isolado

```
1 $ stack new primeiro-projeto simple
2 $ cd primeiro-projeto
3 $ stack setup
4 $ stack build
5 $ stack exec primeiro-projeto
```

Os dois últimos comandos são referentes a compilação do projeto e execução.

O stack cria a seguinte estrutura de diretório:

- LICENSE: informação sobre a licença de uso do software.
- README.md: informações sobre o projeto em formato Markdown.
- Setup.hs: retrocompatibilidade com o sistema cabal.
- primeiro-projeto.cabal: informações das dependências do projeto. Atualizado automaticamente pelo stack.

UFABC

- stack.yaml: parâmetros do projeto
- package.yaml: configurações de compilação e dependências de bibliotecas externas.
- src/Main.hs: arquivo principal do projeto.

UFABC

```haskell
module Main where   -- indica que é o módulo principal

main :: IO ()
main = do                       -- início da função principal
  putStrLn "hello world"        -- imprime hello world
```

UFABC

- Modifique o código `Main.hs` do `primeiro-projeto` criando uma

função triplo que multiplica um valor `x` por `3`.

Modifique a função `main` da seguinte forma para testar:

```haskell
main :: IO ()
main = do
  print (triplo 2)
```

Resposta

UFABC

```haskell
module Main where

triplo x = x * 3

main :: IO ()
main = do
  print (triplo 2)
```

Atividade 2

UFABC

- Crie um programa que utilizando 3 valores (provas, atividades, projeto) calcule a média (numérica) da disciplina. Veja os critérios no site: `http://professor.ufabc.edu.br/~e.francesquini/2019.q2.paradigmas/`