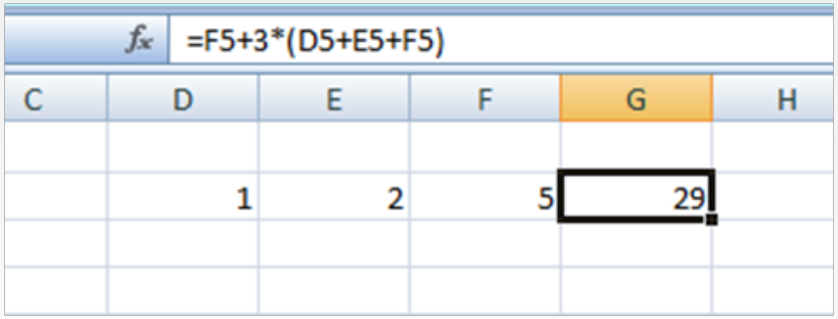
Haskell & Erlang

Introdução a linguagens funcionais

- Programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa;
- O foco esta no alto nível “O que”, ao contrário do baixo nível “Como”;
- Tem sido utilizadas mais utilizadas em aplicações acadêmicas, porém há uso na indústria, como por exemplo o Erlang.

Exemplo 1
Planilha Eletrônica

- Uma planilha eletrônica, onde o valor de uma célula é calculada em função de outras células.



- Não especificamos a ordem em que as células devem ser calculadas, somente sabemos que será executado em uma ordem que respeite as dependências;
- Não dizemos à planilha eletrônica como alocar a memória, esperamos que ela nos presenteie com a aparência de um plano infinito de células e aloque memória somente para as células que estão sendo usadas;
- Especificamos o valor de uma célula por uma expressão, ao invés de especificar por uma sequência de comandos que computam seus valores.

Exemplo 2
Linguagem SQL

- Outro exemplo conhecido de quase linguagem funcional é a SQL.
- Uma consulta SQL é uma expressão envolvendo projeções, seleções, uniões, etc.. A consulta diz que relação deve ser computada, sem dizer como será computada.
- De fato, a consulta pode ser avaliada em uma ordem conveniente.
- As implementações do SQL geralmente apresentam otimizações de consulta, para que seja executada na melhor ordem de avaliação da expressão.

Linguagens Funcionais
Onde são utilizadas na prática?

- A Ericson ulitiza em alguns equipamentos o Erlang;
- Processamento de sinais digitais;
- Reconhecimento de linguagem natural;
- Linguagens de consulta de banco de dados;

Haskell
Historico

- **Haskell** é uma linguagem de programação puramente funcional, de propósito geral, nomeada em homenagem ao lógico Haskell Curry. Como uma linguagem funcional, a estrutura de controle primária é a *função.*
- A primeira versão de Haskell foi definida em 1 de abril de 1990.
  - 1.1 em agosto de 1991;
  - 1.2 em março de 1992;
  - 1.3 em maio de 1996;
  - 1.4 em abril de 1997;
  - Haskell 98, publicado em janeiro de 1999 e que especifica uma versão mínima, estável e portável da linguagem e a biblioteca para ensino. Esse padrão sofreu uma revisão em janeiro de 2003

Exemplo
Quicksort em C

```
void qsort(int a[], int lo, int hi) {
    {
    int h, l, p, t;
    if (lo < hi) {
            l = lo;
            h = hi;
             p = a[hi];
    do {
            while ((l < h) && (a[l] <= p))
                l = l+1;
            while ((h > l) && (a[h] >= p))
                h = h-1;
                if (l < h) {
                    t = a[l];
                    a[l] = a[h];
                    a[h] = t;
                }
    } while (l < h);
    a[hi] = a[l];
    a[l] = p;
    qsort( a, lo, l-1 );
    qsort( a, l+1, hi );
     }
}
```

Exemplo
Quicksort em Haskell

```
qsort [] = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++ qsort (filter (>= x) xs)
```

- Na primeira linha quando tenta ordenar uma lista vazia o resultado é uma outra lista vazia;
- Na segunda linha, para ordenar uma linha chamada de x e o resto chamado de xs, ordena-se os elementos de xs que sao menores que x, ordena os elementos de xs que são maiores ou iguais a x e concatena os resultados (++), onde x é colocado no meio.
  - A função filter filtra a lista através de um predicado ou propriedade. Um predicado é uma função que tem o tipo t->Bool:
  - O ++ é o operador de concatenação de listas.

Erlang
Histórico

- **Erlang** é uma linguagem de programação de uso geral e um sistema para execução. O seu nome é uma homenagem a Agner Krarup Erlang;
- Originalmente de 1987 à 1997 era uma linguagem proprietária da Ericsson, mas foi lançada em código aberto em 1998;
- Foi desenvolvida pela Ericson para suportar distribuição, tolerância a falhas e serem executa em um ambiente de tempo real e interrupto;
- Ela suporta nativamente *hot swapping*, de forma que o código pode ser modificado sem a parada do sistema;
- É uma linguagem interpretada por uma máquina virtual, mas também pode ser compilada;
- A criação de processos é uma tarefa trivial em Erlang, e a comunicação é feita por troca de mensagens.

Exemplo
Quicksort em Erlang

```
-module(quicksort).
 -export([qsort/1]).

 qsort([]) -> [];
 qsort([Pivot|Rest]) ->
     qsort([X || X <- Rest, X < Pivot]) ++ [Pivot] ++ qsort([Y || Y <- Rest, Y >= Pivot]).
```

Haskell
Instalando o Hugs

- Disponível para download gratuitamente em:
  - http://cvs.haskell.org/Hugs/pages/downloading.htm
- Requisitos:
  - Microsoft Windows
  - Debian GNU/Linux
  - Fedora Core/Linux
  - openSUSE/Linux
  - FreeBSD
  - Mac OS X
- Distribuição disponível de setembro de 2006

Haskell
Primeiros programas

- Hugs> "Hello World!"
  - "Hello World!"
- Hugs> 10+5
  - 15
- Hugs> ((3*5)+5)
  - 20
- Existem algumas funções pré-definidas, por exemplo a “reverse”:
  - Main> reverse "Hello World"
    - "dlroW olleH"

Haskell
Arquivos de script

- Embora existam várias funções pré definidas, nas linguagens funcionais busca-se que os usuários criem suas próprias funções.
- As funções dos usuários são definidas em scripts, sendo este script salvo em um arquivo nome_do_arquivo.hs.
- Nos scripts constam definições de funções associando nomes com valores e tipos.
- Podem também ser incluídos comentários, incluindo -- antes dos comentários.
- --Este é um comentário

Haskell
Primeiros programas

Um valor inteiro constante

```
idade :: Int
idade = 17
```

"::" significa possui tipo

- Main>idade
  - 17

Função incrementa (incrementa 1 a um inteiro)

```
incrementa :: Int -> Int
incrementa n = n+1
```

" -> " significa que a função vai de Int para Int

- Main> incrementa 5
  - 6
- Main> incrementa(incrementa 5)
  - 7

Função Mini (retorna o menor valor entre 2 inteiros)

```
mini :: Int -> Int -> Int
mini a b
   |a<=b = a
   |otherwise = b
```

| são os guards que são funções booleanas.
O otherwise é executado quando todos outros guards forem false

- Main> mini 5 6
  - 5

Haskell
Inteiros

- Int é o tipo dos números inteiros em java;
- Apresenta os operadores de inteiros (*, +, - (negação e subtração));
- Funções (div, mod, abs, negate);
- As funções podem ser usadas como operadores e os operadores como funções;
  - Main> (+)7 3
    - 10
  - Main> 15 `mod` 3
    - 0

Note que o operador deve estar entre parênteses

Note que as funções devem estar entre crases `

- Definindo seus próprios operadores

```
(&&&) :: Int -> Int -> Int

a &&& b
   | a < b = a
   | otherwise = b
```

```
Hugs> 5 &&& 6
5
```

Haskell
Booleanos

- Bool é o tipo dos boobleanos e pode ter valor True ou False;
- Possui os operadores && (and), || (or), not (negation);

Haskell
Definindo tipos personalizados

- Para definir tipos próprios basta usar a declaração **data.**
- **Exemplo**
  - data Bool = True | False
  - data Cor = Azul | Verde | Vermelho | Amarelo
  - **Declaração de um tipo Point que contém uma tupla e é um tipo polimórfico.**
  - data Point = Pt a a

Haskell
Caracteres e Strings

- O tipo Char é composto por caracteres, digitos e caracteres especiais:
  - Devem ser escritos entre aspas simples ‘;
  - Os caracteres especiais correspondem à \t, \n, ‘\’’, ‘\”’ e ‘\\’
  - Os caracteres podem ser representados pelo seu número da tabela ASCII;

```
Main> '\100'
   'd‘
Main> '8'
   '8‘
```

A \ define que se está referenciando o caractere especial

Esta referenciando o caractere 8

- O tipo string é composto por palavras:
  - Devem ser escritas entre aspas duplas”;
  - As strings são sinônimos de uma coleção de caracteres

```
Main> "Haskell" == ['H', 'a', 's', 'k', 'e', 'l', 'l']
   True
```

Haskell
Números de ponto flutuante

- Números fracionários são tratados como ponto flutuante;
- Pode-se escrever os números:
- 231.6e-2
- 231.61 ´ 10-2
- 2.3161.
- O tipo Float além de aceitar os operadores (+, - , *, ^, = =, /=, <=,>=, <, >) dos inteiros, possui algumas funções próprias:
  - /, **, cos, sin, tan, log, sqrt, entre outros.

Haskell
Tuplas

- Agregação de um ou mais compontentes, que inclusive podem ser de tipos diferentes:

```
type Nome = String

type Idade = Int

verIdade :: (Nome, Idade) -> Idade
verIdade (a,b) = b
```

- Main> verIdade ("Joao", 18)
- 18

Haskell
Funções recursivas

- Função recursiva é a função que chama ela mesma. Os loops das linguagens imperativas podem ser simulados por uma função recursiva.

```
fatorial :: Int -> Int
fatorial 0 = 1
fatorial n = n * fatorial (n-1)
```

  - Main> fatorial 3
    - 6

Exercícios
Sequência de Fibonacci

- Descreve o crescimento de uma população de coelhos. Os números descrevem o número de casais em uma população de coelhos depois de *n* meses se for suposto que:
  - no primeiro mês nasce apenas um casal,
  - casais amadurecem sexualmente (e reproduzem-se) apenas após o segundo mês de vida,
  - não há problemas genéticos no cruzamento consanguíneo,
  - todos os meses, cada casal fértil dá a luz a um novo casal, e
  - os coelhos nunca morrem.

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1\,. \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

Exercícios:
Implementação Fibonacci

```
fib :: Int -> Int
fib n
   |n == 0 = 0
   |n == 1 = 1
   |otherwise = fib(n-1) + fib(n-2)
```

OU

```
fib :: Int -> Int
fib n
   |n == 0 = 0
   |n == 1 = 1
   |n > 1 = fib(n-1) + fib(n-2)
```

OU

```
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = fib(n-1) + fib(n-2)
```

Haskell
Listas

- Podem ser de diferentes tipos, por exemplo, lista de inteiros, lista de caracteres, lista de booleanos, lista de strings, etc.;
- As listas também podem conter listas (listas de lista), tuplas e ainda funções, desde que sejam do mesmo tipo.
- [5,6,8,4,7,55] :: Int
- [‘a’, ‘b’, ‘c’] :: Char
- [True, True, False] :: Bool
- [[1,2,3], [4,5,6], [7,8,9]] :: [[Int]]
- [(1,’a’),(2,’b’),(3,’c’)] ::[(Int, Char)]

Haskell
Listas

- Geração automática de listas
  - O Haskell permite que se gere uma lista automaticamente, por exemplo:
    - Main>[1..10]

      [1,2,3,4,5,6,7,8,9,10]
    - Main>[10..1]

      []
    - Main>[2,4..10]

      [2,4,6,8,10]
    - Main>['a'..'z']

      "abcdefghijklmnopqrstuvwxyz"

Haskell
Listas

- Operadores
  - Operador de construção “ : ”
    - O operador “ : “ é utilizado para a construção de listas;

    [1,2,3,4] = 1:2:3:4:[]
  - Operador de concatenação “++”
    - O operador de concatenação ++ é utilizado para concatenar listas:

    Main>[1,2,3]++[4,5,6]++[7,8,9]

    [1,2,3,4,5,6,7,8,9]

Haskell
Funções sobre listas

- Uma lista é representada por um “head” e um ”tail” (x:xs), onde x é o primeiro elemento da lista e xs é o resto da lista;
- Na lista [1,2,3], o head é 1 e o tail é o [2,3];
- Exemplo
  - Função SomaLista
    - somaLista [] = 0
    - somaLista (x:xs) = x + somaLista xs
  - Main>somaLista [1..10]

    55

Haskell
Compreensão de listas

- Notação para geração de novas listas baseadas em listas já existentes
- Notação próxima à notação matemática para descrição de conjuntos
- Pode ocorrer em qualquer lugar, onde poderia ocorrer uma lista
- Uma compreensão de listas é equivalente a uma lista
- $S = \{x \mid x \in N, x^2 > 3\}$
  - Main> [x | x←[0..10], x^2>3]

    [2,3,4,5,6,7,8,9,10]

Haskell
Compreensão de listas

- ***[2 * b | b <- lista, b < 10]***

- Gerador

  b* <- *lista (***a*** representa cada elemento da ***lista***)

- Sobre cada elemento da lista
  - **Expressões**
    - ***2 * b***
  - **Filtros**
    - ***b < 10***

Compreensão de listas

Exemplos:

- Considerando a lista lista = [1, 2, 5, 15, 20]
- Um somente filtro

```
[b | b <- lista, b < 10]
Resultado: [1, 2, 5]
```

- Operação e filtro

```
[2 * b | b <- lista, b < 10]
Resultado: [2, 4, 10]
```

- Produto cartesiano

```
[ (x,y) | x <- [1,2,3], y <- [2,3,5] ]
Resultado:[(1,2)(2,3)(3,5)]
```

Compreensão de listas

- A maioria das definições sobre listas se encaixam em três casos:
  - *folding, que é a* colocação de um operador entre os elementos de uma lista;
  - *filtering, que significa filtrar* alguns elementos e;
  - *mapping que é a aplicação de funções a todos os elementos da lista.*
- *Os* outros casos são combinações destes três, ou recursões primitivas.
- Os 3 casos citados são resolvidos pelas funções primitivas:
  - foldr1
  - map
  - filter

Funções pré-definidas para listas

- foldr1:
  - Adiciona um operador entre os elementos de uma lista:
    - Main> foldr1 (+) [1..10]

      55
- map
  - Aplica uma função a cada elemento da lista:
    - Exemplo:
      - duplica :: Int -> Int
      - duplica x = 2 * x

    - Main> map duplica [1..10]

      [2,4,6,8,10,12,14,16,18,20]

Haskell
Funções pré-definidas para listas

- filter:
  - Filtra uma lista através de uma propriedade ou predicado:
    - Exemplo:
      - Main> filter (>5) [1..10]

        [6,7,8,9,10]

Exercício
Listas

- Criar uma lista de tuplas conforme segue abaixo:
- [(“Corsa”, “GM”, 2005), (“Fiesta”, “Ford”, 2008), (“Celta”, “GM”, 2003), (“Palio”, “Fiat”, 2010), (“Uno”, “Fiat”, 1995), (“Ka”, “Ford”, 2010)]
- Crie um arquivo de script .hs
- Defina o tipo da lista.
- Crie a lista
- Escreva operações sobre a lista, como por exemplo:
  - Uma compreensão de listas para listar o nome dos carros da GM.
  - Listar o nome dos carros do ano de 2010
  - Crie novas consultas, aplique operações sobre esta lista e demais alterações sobre ela para fixar melhor os conceitos.
  - Sugiro utilizar a referência [1] como base para consulta.

Referências

- [1] **http://www.macs.hw.ac.uk/~dubois/ProgramacaoHaskell.pdf**
- [2] http://www.marcosrodrigues6.hpg.ig.com.br/cap1.htm
- [3] http://www.cin.ufpe.br/~alms/pdf/JogosEducativosLinguagensFuncionais.pdf
- [4] http://caioariede.com/2009/aprendendo-erlang-parte-1
- [5] http://www.haskell.org/haskellwiki/Introduction
- [6] http://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_funcional#
- [7] http://www.erlang.se/publications/bjarnelic.pdf
- [8] http://www.haskell.org/tutorial/intro.html