

Programação Funcional em Haskell

José Romildo Malaquias

BCC222: Programação Funcional

Universidade Federal de Ouro Preto
Departamento de Computação

27 de setembro de 2023

1	Paradigmas de Programação	1-1
1.1	Paradigmas de programação	1-1
1.2	Técnicas e paradigmas de programação	1-2
1.3	Categorias: programação imperativa e declarativa	1-2
1.3.1	Programação imperativa	1-2
1.3.2	Programação declarativa	1-2
1.4	Programação funcional	1-3
1.4.1	Exemplo: quick sort em C	1-3
1.4.2	Exemplo: quick sort em Haskell	1-3
1.5	A Crise do Software	1-4
1.6	Algumas características de Haskell	1-4
1.7	Antecedentes históricos	1-4
1.8	Algumas empresas que usam Haskell	1-7
1.9	Curso online de Haskell	1-7
2	Ambiente de Desenvolvimento Haskell	2-1
2.1	Haskell	2-1
2.2	Instalação do ambiente de desenvolvimento	2-2
2.2.1	Instalação das ferramentas de desenvolvimento em Haskell	2-2
2.3	O ambiente interativo GHCi	2-2
2.4	Bibliotecas	2-6
3	Expressões e Definições	3-1
3.1	Constantes	3-1
3.2	Aplicação de função	3-2
3.3	Nomeando valores	3-6
3.4	Avaliando expressões	3-7
3.5	Definindo variáveis e funções	3-8
3.6	Comentários	3-9
3.7	Definições locais em equações	3-10
3.8	Regra de <i>layout</i>	3-11
3.9	Comandos úteis do GHCi	3-12
3.10	Exercícios	3-12
3.11	Soluções	3-15
4	Tipos de Dados	4-1
4.1	Tipos	4-1
4.2	Alguns tipos básicos	4-1
4.3	Tipos função	4-3
4.4	Checação de tipos	4-3
4.5	Assinatura de tipo em definições	4-4
4.6	Consulta do tipo de uma expressão no GHCi	4-4
4.7	Soluções	4-6

5	Estruturas de dados básicas	5-1
5.1	Tuplas	5-1
5.2	Listas	5-2
5.2.1	Progressão aritmética	5-4
5.3	Strings	5-5
5.4	Valores opcionais	5-5
5.5	Exercícios	5-6
5.6	Soluções	5-7
6	Polimorfismo Paramétrico	6-1
6.1	Operação sobre vários tipos de dados	6-1
6.2	Variáveis de tipo	6-1
6.3	Valor polimórfico	6-2
6.4	Instanciação de variáveis de tipo	6-2
6.5	Algumas funções polimórficas predefinidas	6-2
6.6	Exercícios	6-3
6.7	Soluções	6-4
7	Sobrecarga	7-1
7.1	Sobrecarga	7-1
7.2	Algumas classes de tipo pré-definidas	7-2
7.2.1	Eq	7-2
7.2.2	Ord	7-3
7.2.3	Enum	7-3
7.2.4	Bounded	7-3
7.2.5	Show	7-4
7.2.6	Read	7-4
7.2.7	Num	7-4
7.2.8	Real	7-4
7.2.9	Integral	7-5
7.2.10	Fractional	7-5
7.2.11	Floating	7-5
7.2.12	RealFrac	7-6
7.2.13	RealFloat	7-6
7.3	Sobrecarga de literais	7-7
7.4	Conversão entre tipos numéricos	7-7
7.5	Inferência de tipos	7-8
7.6	Dicas e Sugestões	7-8
7.7	Exercícios	7-9
7.8	Soluções	7-11
8	Expressão Condicional	8-1
8.1	Expressão condicional	8-1
8.2	Definição de função com expressão condicional	8-2
8.3	Equações com guardas	8-3
8.4	Definições locais e guardas	8-5
8.5	Exercícios	8-7
8.6	Soluções	8-9

9	Programas Interativos	9-1
9.1	Interação com o <i>mundo</i>	9-1
9.1.1	Programas interativos	9-1
9.1.2	Linguagens puras	9-2
9.1.3	O mundo	9-2
9.1.4	Modificando o mundo	9-3
9.1.5	Ações de entrada e saída	9-3
9.2	O tipo <code>unit</code>	9-3
9.3	Ações de saída padrão	9-3
9.4	Ações de entrada padrão	9-4
9.5	Programa em Haskell	9-5
9.6	Combinando ações de entrada e saída	9-6
9.7	Exemplos de programas interativos	9-7
9.8	Saída bufferizada	9-9
9.9	Mais exemplos de programas interativos	9-11
9.10	Exercícios	9-12
9.11	Soluções	9-18
10	Funções Recursivas	10-1
10.1	Recursividade	10-1
10.2	Recursividade mútua	10-5
10.3	Recursividade de cauda	10-6
10.4	Vantagens da recursividade	10-9
10.5	Exercícios	10-9
10.6	Soluções	10-11
11	Ações de E/S Recursivas	11-1
11.1	A função <code>return</code>	11-1
11.2	Exemplo: exibir uma sequência	11-1
11.3	Exemplo: somar uma sequência	11-1
11.4	Problemas	11-3
11.5	Soluções	11-6
12	Casamento de Padrão	12-1
12.1	Casamento de padrão	12-1
12.1.1	Casamento de padrão	12-1
12.1.2	Padrão constante	12-2
12.1.3	Padrão variável	12-2
12.1.4	Padrão curinga	12-2
12.1.5	Padrão tupla	12-3
12.1.6	Padrões lista	12-3
12.1.7	Padrão lista na notação especial	12-4
12.2	Definição de função usando padrões	12-5
12.2.1	Definindo funções com casamento de padrão	12-5
12.3	Casamento de padrão em definições	12-9
12.4	Problema: validação de números de cartão de crédito	12-10
12.5	Problema: torres de Hanoi	12-12
12.6	Soluções	12-14

13	Expressão de Seleção Múltipla	13-1
13.1	Expressão case	13-1
13.2	Forma e regras de tipo da expressão case	13-1
13.3	Regra de <i>layout</i> para a expressão case	13-2
13.4	Avaliação de expressões case	13-2
13.5	Exemplos de expressões case	13-3
13.6	Expressão case com guardas	13-5
13.7	Soluções	13-7
14	Valores Aleatórios	14-1
14.1	Instalação do pacote random	14-1
14.2	Valores aleatórios	14-1
14.3	Jogo: adivinha o número	14-2
14.4	Soluções	14-9
15	Expressão Lambda	15-1
15.1	Valores de primeira classe	15-2
15.1.1	Valores de primeira classe	15-2
15.1.2	Valores de primeira classe: Literais	15-2
15.1.3	Valores de primeira classe: Variáveis	15-2
15.1.4	Valores de primeira classe: Argumentos	15-3
15.1.5	Valores de primeira classe: Resultado	15-3
15.1.6	Valores de primeira classe: Componentes	15-3
15.2	Expressão lambda	15-3
15.2.1	Expressões lambda	15-3
15.2.2	Exemplos de expressões lambda	15-4
15.2.3	Uso de expressões lambda	15-4
15.2.4	Exercícios	15-5
15.3	Aplicação parcial de funções	15-6
15.3.1	Aplicação parcial de funções	15-6
15.3.2	Aplicação parcial de funções: exemplos	15-6
15.4	<i>Currying</i>	15-8
15.4.1	Funções <i>curried</i>	15-8
15.4.2	Por que <i>currying</i> é útil?	15-8
15.4.3	Convenções sobre <i>currying</i>	15-8
15.5	Seções de operadores	15-9
15.5.1	Operadores	15-9
15.5.2	Seções de operadores	15-10
15.6	Utilidade de expressões lambda	15-11
15.6.1	Por que seções são úteis?	15-11
15.6.2	Utilidade de expressões lambda	15-12
15.6.3	Exercícios	15-14
15.7	Soluções	15-15
16	Funções de Ordem Superior	16-1
16.1	Funções de Ordem Superior	16-1
16.2	Um operador para aplicação de função	16-1
16.3	Composição de funções	16-2
16.4	A função filter	16-3
16.5	A função map	16-3
16.6	A função zipWith	16-4
16.7	As funções foldl e foldr , foldl1 e foldr1	16-4

16.7.1	<code>foldl</code>	16-4
16.7.2	<code>foldr</code>	16-5
16.7.3	<code>foldl1</code>	16-5
16.7.4	<code>foldr1</code>	16-6
16.8	<i>List comprehension</i>	16-6
16.8.1	<i>List comprehension</i>	16-6
16.8.2	<i>List comprehension</i> e funções de ordem superior	16-7
16.9	Cupom fiscal do supermercado	16-8
16.10	Soluções	16-13
17	Argumentos da Linha de Comando e Arquivos	17-1
17.1	Argumentos da linha de comando	17-1
17.2	Encerrando o programa explicitamente	17-2
17.3	Formatando dados com a função <code>printf</code>	17-4
17.4	Arquivos	17-5
17.5	As funções <code>lines</code> e <code>unlines</code> , e <code>words</code> e <code>unwords</code>	17-6
17.6	Exemplo: processar notas em arquivo	17-7
17.7	Problemas	17-8
17.8	Soluções	17-11
18	Tipos Algébricos	18-1
18.1	Novos tipos de dados	18-1
18.2	Tipos algébricos	18-2
18.3	Exemplo: formas geométricas	18-2
18.4	Exemplo: sentido de movimento	18-3
18.5	Exemplo: cor	18-5
18.6	Exemplo: coordenadas cartesianas	18-5
18.7	Exemplo: horário	18-5
18.8	Exemplo: booleanos	18-6
18.9	Exemplo: listas	18-6
18.10	Exercícios básicos	18-7
18.11	Números naturais	18-8
18.12	Árvores binárias	18-9
18.13	O construtor de tipo <code>Maybe</code>	18-9
18.14	Exercício: lógica proposicional	18-10
18.15	Soluções	18-16
19	Classes de Tipos	19-1
19.1	Polimorfismo <i>ad hoc</i> (sobrecarga)	19-1
19.2	Tipos qualificados	19-2
19.3	Classes e Instâncias	19-2
19.4	Tipo principal	19-3
19.5	Definição padrão	19-3
19.6	Exemplos de instâncias	19-4
19.7	Instâncias com restrições	19-4
19.8	Derivação de instâncias	19-5
19.8.1	Herança	19-5
19.9	Alguma classes do prelúdio	19-5
19.9.1	A classe <code>Show</code>	19-5
19.9.2	A classe <code>Eq</code>	19-6
19.9.3	A classe <code>Ord</code>	19-6
19.9.4	A classe <code>Enum</code>	19-6

19.9.5 A classe Num	19-7
19.10 Exercícios	19-8
19.11 Soluções	19-11
20 Valores em um Contexto	20-1
20.1 Valores encapsulados	20-1
20.2 Aplicação de função	20-2
20.3 Funtores	20-2
20.4 Funtores aplicativos	20-3
20.5 Mônadas	20-3
20.6 Exemplo: expressões aritméticas	20-5
20.7 Exemplo: geração de histórico	20-7
21 Parsers	21-1
21.1 Parsers	21-1
21.2 Parsers como funções	21-1
21.3 Soluções	21-3

1 Paradigmas de Programação

Resumo

Ao desenvolver uma aplicação o programador segue uma visão de como o programa será executado, norteando a estruturação do seu código. Isto é o que chamamos de **paradigma de programação**.

Existem alguns paradigmas muito usados como a programação procedimental, a programação orientada a objetos, a programação funcional e a programação lógica. Cada um deles tem uma visão diferente da estrutura e execução dos programas.

Sumário

1.1 Paradigmas de programação	1-1
1.2 Técnicas e paradigmas de programação	1-2
1.3 Categorias: programação imperativa e declarativa	1-2
1.3.1 Programação imperativa	1-2
1.3.2 Programação declarativa	1-2
1.4 Programação funcional	1-3
1.4.1 Exemplo: quick sort em C	1-3
1.4.2 Exemplo: quick sort em Haskell	1-3
1.5 A Crise do Software	1-4
1.6 Algumas características de Haskell	1-4
1.7 Antecedentes históricos	1-4
1.8 Algumas empresas que usam Haskell	1-7
1.9 Curso online de Haskell	1-7

1.1 Paradigmas de programação

- Um **paradigma de programação** fornece e determina a visão que o programador possui sobre a estruturação e a execução do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
- Diferentes linguagens de programação propõem diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.

- Por exemplo:
 - Smalltalk, Eiffel e Java suportam o paradigma orientado a objetos.
 - Haskell e Clean suportam o paradigma funcional.
 - OCaml, LISP, Scala, Perl, Python e C++ suportam múltiplos paradigmas.

1.2 Técnicas e paradigmas de programação

- Geralmente os paradigmas de programação são diferenciados pelas técnicas de programação que **permitem** ou **proíbem**.
- Por exemplo, a **programação estruturada** não permite o uso de *goto*.
- Esse é um dos motivos pelos quais novos paradigmas são considerados mais rígidos que estilos tradicionais.
- Apesar disso, evitar certos tipos de técnicas pode facilitar a prova de correção de um sistema, podendo até mesmo facilitar o desenvolvimento de algoritmos.
- O relacionamento entre paradigmas de programação e linguagens de programação pode ser complexo pelo fato de linguagens de programação poderem suportar mais de um paradigma.

1.3 Categorias: programação imperativa e declarativa

1.3.1 Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que mudam o **estado** (**variáveis**) de um programa, enfatizando *como* resolver um problema.
- Muito parecidos com o comportamento imperativo das linguagens naturais que expressam ordens, programas imperativos são uma sequência de comandos para o computador executar.
- O nome do paradigma, *imperativo*, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, etc

1.3.2 Programação declarativa

- Descreve *o que* o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos *resultados*, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, LISP, etc, e
 - **lógico**: Prolog, etc.

1.4 Programação funcional

- **Programação funcional** é um paradigma de programação que descreve uma computação como uma *expressão* a ser avaliada.
- A principal forma de estruturar o programa é pela definição e aplicação de *funções*.

1.4.1 Exemplo: quick sort em C

```
// To sort array a[] of size n: qsort(a,0,n-1)

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);
    }
}
```

1.4.2 Exemplo: quick sort em Haskell

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

Observações:

- `[]` denota a lista vazia.
- `x:xs` denota uma lista não vazia cuja cabeça é `x` e cuja cauda é `xs`.
- Uma lista pode ser escrita enumerando os seus elementos separados por vírgula e colocados entre colchetes.
- A sintaxe para aplicação de função consiste em escrever a função seguida dos argumentos, separados por espaços, como em `max 10 (2+x)`.
- A função `filter` seleciona os elementos de uma lista que satisfazem uma determinada propriedade.
- `(<x)` e `(>x)` são funções que verificam se o seu argumento é menor ou maior, respectivamente, do que `x`. São funções anônimas construídas pela aplicação parcial dos operadores `<` e `>`.
- O operador `++` concatena duas listas.

1.5 A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma clara, concisa, e com um alto nível de abstração;
 - suportam componentes de software reutilizáveis;
 - incentivam o uso de verificação formal;
 - permitem prototipagem rápida;
 - fornecem poderosas ferramentas de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o *tamanho* e a *complexidade* dos programas de computador modernos
 - o *tempo* e o *custo* de desenvolvimento do programas
 - *confiança* de que os programas já concluídos funcionam *corretamente*



As **linguagens funcionais** oferecem um quadro particularmente elegante para abordar estes objetivos.

1.6 Algumas características de Haskell

- Programas são concisos
- Tipagem estática
- Sistema de tipos poderoso
- Tipos e funções recursivas
- Funções de ordem superior
- Linguagem pura (declarativa)
- Avaliação *lazy*
- Maior facilidade de raciocínio sobre programas

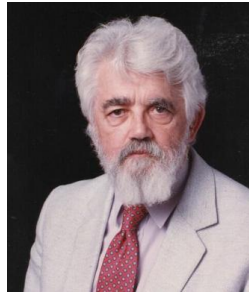
1.7 Antecedentes históricos

- Década de 1930:



Alonzo Church desenvolve o **cálculo lambda**, uma teoria de funções simples, mas poderosa.

- Década de 1950:



John McCarthy desenvolve **Lisp**, a primeira linguagem funcional, com algumas influências do **cálculo lambda**, mas mantendo as atribuições de variáveis.

- Década de 1960:



Peter Landin desenvolve **ISWIM**, a primeira linguagem funcional pura, baseada fortemente no **cálculo lambda**, sem atribuições.

- Década de 1970:



John Backus desenvolve **FP**, uma linguagem funcional que enfatiza funções de ordem superior e raciocínio sobre programas.

- Década de 1970:



Robin Milner e outros desenvolvem **ML**, a primeira linguagem funcional moderna, que introduziu a inferência de tipos e tipos polimórficos.

- Décadas de 1970 e 1980:



David Turner desenvolve uma série de linguagens funcionais com *avaliação lazy*, culminando com o sistema **Miranda**.

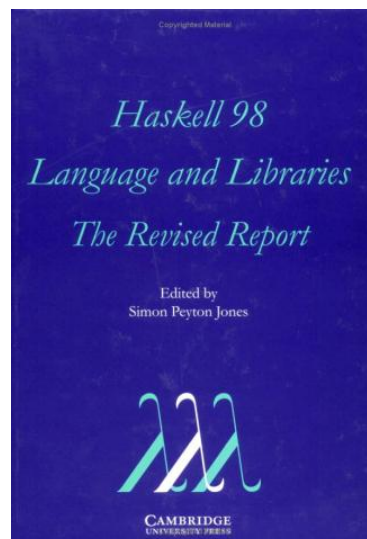
- 1987:

Haskell

A Purely Functional Language

Um comitê internacional de pesquisadores inicia o desenvolvimento de **Haskell**, uma linguagem funcional *lazy* padrão.

- 2003:



O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem Haskell.

- 2009:



O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem Haskell.

1.8 Algumas empresas que usam Haskell

- Exemplos de empresas que usam Haskell:
 - **ABN AMRO** análise de riscos financeiros
 - **AT&T** automatização de processamento de formulários
 - **Bank of America Merrill Lynch** transformação de dados
 - **Bump** servidores baseados em Haskell
 - **Facebook** manipulação da base de código PHP
 - **Google** infra-estrutura interna de TI
 - **MITRE** análise de protocolos de criptografia
 - **NVIDIA** ferramentas usadas internamente
 - **Qualcomm, Inc** geração de interfaces de programação para Lua
 - **The New York Times** processamento de imagens
- Para maiores detalhes visite a página *Haskell na indústria* em http://www.haskell.org/haskellwiki/Haskell_in_industry.

1.9 Curso online de Haskell



- Functional Programming in Haskell
- Universidade de Glasgow
- Início: 19 de setembro de 2016
- Duração: 6 semanas
- Dedicção: 4 horas por semana
- <https://www.futurelearn.com/courses/functional-programming-haskell>

