Programação Funcional com Haskell

Paulo Torrens

paulotorrens@gnu.org

Departamento de Ciência da Computação
Centro de Ciências e Tecnológias
Universidade do Estado de Santa Catarina

2019/02

Introdução

- A maioria das linguagens de programação populares se enquadram no paradigma imperativo
  - Em linguagens imperativas, o programador controla o estado da aplicação explicitamente através de comandos, que podem ter efeitos colaterais arbitrários
  - Efeitos colaterais são as ações executadas além do retorno da função e que manipulam o estado do programa: alterar a memória, escrever um arquivo, imprimir algo no terminal, etc
  - Dentre os possíveis comandos também temos atribuições a variáveis e estruturas de fluxo de controle (`while`, `for`, etc), que especificam como executar um algoritmo
  - Exemplos: C, C++, Objective-C, Fortran, Pascal, Java, JavaScript, Python, Swift, Rust, etc

Introdução

- A maioria das linguagens de programação populares se enquadram no paradigma imperativo
  - Em linguagens imperativas, o programador controla o estado da aplicação explicitamente através de comandos, que podem ter efeitos colaterais arbitrários
  - Efeitos colaterais são as ações executadas além do retorno da função e que manipulam o estado do programa: alterar a memória, escrever um arquivo, imprimir algo no terminal, etc
  - Dentre os possíveis comandos também temos atribuições a variáveis e estruturas de fluxo de controle (`while`, `for`, etc), que especificam como executar um algoritmo
  - Exemplos: C, C++, Objective-C, Fortran, Pascal, Java, JavaScript, Python, Swift, Rust, etc

Introdução

- A maioria das linguagens de programação populares se enquadram no paradigma imperativo
  - Em linguagens imperativas, o programador controla o estado da aplicação explicitamente através de comandos, que podem ter efeitos colaterais arbitrários
  - Efeitos colaterais são as ações executadas além do retorno da função e que manipulam o estado do programa: alterar a memória, escrever um arquivo, imprimir algo no terminal, etc
  - Dentre os possíveis comandos também temos atribuições a variáveis e estruturas de fluxo de controle (`while`, `for`, etc), que especificam como executar um algoritmo
  - Exemplos: C, C++, Objective-C, Fortran, Pascal, Java, JavaScript, Python, Swift, Rust, etc

Introdução

- A maioria das linguagens de programação populares se enquadram no paradigma imperativo
  - Em linguagens imperativas, o programador controla o estado da aplicação explicitamente através de comandos, que podem ter efeitos colaterais arbitrários
  - Efeitos colaterais são as ações executadas além do retorno da função e que manipulam o estado do programa: alterar a memória, escrever um arquivo, imprimir algo no terminal, etc
  - Dentre os possíveis comandos também temos atribuições a variáveis e estruturas de fluxo de controle (`while`, `for`, etc), que especificam como executar um algoritmo
  - Exemplos: C, C++, Objective-C, Fortran, Pascal, Java, JavaScript, Python, Swift, Rust, etc

Introdução

- A maioria das linguagens de programação populares se enquadram no paradigma imperativo
  - Em linguagens imperativas, o programador controla o estado da aplicação explicitamente através de comandos, que podem ter efeitos colaterais arbitrários
  - Efeitos colaterais são as ações executadas além do retorno da função e que manipulam o estado do programa: alterar a memória, escrever um arquivo, imprimir algo no terminal, etc
  - Dentre os possíveis comandos também temos atribuições a variáveis e estruturas de fluxo de controle (`while`, `for`, etc), que especificam como executar um algoritmo
  - Exemplos: C, C++, Objective-C, Fortran, Pascal, Java, JavaScript, Python, Swift, Rust, etc

Introdução

```java
// Java
int my_number = 0;

int getNumber() {
  System.out.println("Getting a number!");
  return my_number++;
}

void showNumber() {
  System.out.println("Got: " + getNumber());
  System.out.println("Got: " + getNumber());
  System.out.println("Got: " + getNumber());
}
```

Introdução

- Em contraste às linguagens imperativas, existem as linguagens que se enquadram no paradigma declarativo
  - Linguagens declarativas desencorajam ou proíbem o uso de código que manipule explicitamente estado ou com efeitos colaterais
  - Funções são definidas apenas em respeito aos seus parâmetros de entrada e ao seu resultado
- Dentro do paradigma declarativo, temos o paradigma funcional, onde funções são tratadas como declarações matemáticas
  - Essas linguagens priorizam a manipulação de funções
  - Funções podem ser passadas como argumentos e podem ser retornadas como resultado de outras funções
  - Exemplos: Common Lisp, Scheme, Standard ML, Erlang, Elixir, Haskell, etc

Introdução

- Em contraste às linguagens imperativas, existem as linguagens que se enquadram no paradigma declarativo
  - Linguagens declarativas desencorajam ou proíbem o uso de código que manipule explicitamente estado ou com efeitos colaterais
  - Funções são definidas apenas em respeito aos seus parâmetros de entrada e ao seu resultado
- Dentro do paradigma declarativo, temos o paradigma funcional, onde funções são tratadas como declarações matemáticas
  - Essas linguagens priorizam a manipulação de funções
  - Funções podem ser passadas como argumentos e podem ser retornadas como resultado de outras funções
  - Exemplos: Common Lisp, Scheme, Standard ML, Erlang, Elixir, Haskell, etc

Introdução

- Em contraste às linguagens imperativas, existem as linguagens que se enquadram no paradigma declarativo
  - Linguagens declarativas desencorajam ou proíbem o uso de código que manipule explicitamente estado ou com efeitos colaterais
  - Funções são definidas apenas em respeito aos seus parâmetros de entrada e ao seu resultado
- Dentro do paradigma declarativo, temos o paradigma funcional, onde funções são tratadas como declarações matemáticas
  - Essas linguagens priorizam a manipulação de funções
  - Funções podem ser passadas como argumentos e podem ser retornadas como resultado de outras funções
  - Exemplos: Common Lisp, Scheme, Standard ML, Erlang, Elixir, Haskell, etc

Introdução

- Em contraste às linguagens imperativas, existem as linguagens que se enquadram no paradigma declarativo
  - Linguagens declarativas desencorajam ou proíbem o uso de código que manipule explicitamente estado ou com efeitos colaterais
  - Funções são definidas apenas em respeito aos seus parâmetros de entrada e ao seu resultado
- Dentro do paradigma declarativo, temos o paradigma funcional, onde funções são tratadas como declarações matemáticas
  - Essas linguagens priorizam a manipulação de funções
  - Funções podem ser passadas como argumentos e podem ser retornadas como resultado de outras funções
  - Exemplos: Common Lisp, Scheme, Standard ML, Erlang, Elixir, Haskell, etc

Introdução

- Em contraste às linguagens imperativas, existem as linguagens que se enquadram no paradigma declarativo
  - Linguagens declarativas desencorajam ou proíbem o uso de código que manipule explicitamente estado ou com efeitos colaterais
  - Funções são definidas apenas em respeito aos seus parâmetros de entrada e ao seu resultado
- Dentro do paradigma declarativo, temos o paradigma funcional, onde funções são tratadas como declarações matemáticas
  - Essas linguagens priorizam a manipulação de funções
  - Funções podem ser passadas como argumentos e podem ser retornadas como resultado de outras funções
  - Exemplos: Common Lisp, Scheme, Standard ML, Erlang, Elixir, Haskell, etc

Introdução

- Em contraste às linguagens imperativas, existem as linguagens que se enquadram no paradigma declarativo
  - Linguagens declarativas desencorajam ou proíbem o uso de código que manipule explicitamente estado ou com efeitos colaterais
  - Funções são definidas apenas em respeito aos seus parâmetros de entrada e ao seu resultado
- Dentro do paradigma declarativo, temos o paradigma funcional, onde funções são tratadas como declarações matemáticas
  - Essas linguagens priorizam a manipulação de funções
  - Funções podem ser passadas como argumentos e podem ser retornadas como resultado de outras funções
  - Exemplos: Common Lisp, Scheme, Standard ML, Erlang, Elixir, Haskell, etc

Introdução

- Em contraste às linguagens imperativas, existem as linguagens que se enquadram no paradigma declarativo
  - Linguagens declarativas desencorajam ou proíbem o uso de código que manipule explicitamente estado ou com efeitos colaterais
  - Funções são definidas apenas em respeito aos seus parâmetros de entrada e ao seu resultado
- Dentro do paradigma declarativo, temos o paradigma funcional, onde funções são tratadas como declarações matemáticas
  - Essas linguagens priorizam a manipulação de funções
  - Funções podem ser passadas como argumentos e podem ser retornadas como resultado de outras funções
  - Exemplos: Common Lisp, Scheme, Standard ML, Erlang, Elixir, Haskell, etc

Introdução

```
1 (* Standard ML *)
2 fun fib 0 = 0
3   | fib 1 = 1
4   | fib n = fib (n - 1) + fib (n - 2)
```

Exercícios de Recursão

1. Uma função que some os números $1 + 2 + 3 + \ldots + n$.

```
int recsum(int n)
```

2. Uma função que encontre o menor elemento de uma lista.

```
int findmin(int list[], int len)
```

3. Uma função que encontre o maior elemento de uma lista.

```
int findmax(int list[], int len)
```

4. Uma função que verifique se uma palavra é um palíndromo.

```
_Bool palin(char word[], int len)
```

Exercícios de Recursão

```c
// C
int recsum(int n) {
  if(n == 0) {
    return 0;
  }
  return n + recsum(n - 1);
}
```

Exercícios de Recursão

```python
# Python
def recsum(n):
    if n == 0:
        return 0
    return n + recsum(n - 1)
```

Exercícios de Recursão

```c
// C
int findmin(int list[], int len) {
  if(len == 1) {
    return list[0];
  }

  int top = list[len - 1];
  int aux = findmin(list, len - 1);
  if(top > aux) {
    return aux;
  } else {
    return top;
  }
}
```

Exercícios de Recursão

```python
# Python
def findmax(items):
  if len(items) == 1:
    return items[0]
  else:
    top = items[0]
    aux = findmax(items[1:])
    if top < aux:
      return aux
    else:
      return top
```

Exercícios de Recursão

```c
// C
_Bool palin(char word[], int len) {
  if(len < 2) {
    return 1;
  }
  if(word[0] != word[len - 1]) {
    return 0;
  }
  return palin(word + 1, len - 2);
}
```

Exercícios de Recursão

```python
# Python
def palin(array):
  if len(array) < 2:
    return True
  if array[0] != array[len(array) - 1]:
    return False
  return palin(array[1:-1])
```

Haskell

"Haskell é uma linguagem de programação puramente funcional. Em linguagens imperativas você faz as coisas dando ao computador uma sequência de tarefas e então as executando. Enquanto você as executa, o estado pode ser alterado. Por exemplo, você seta a variável a para 5 e então faz outras coisas e então a seta para algo diferente. Você tem estruturas de fluxo de controle para fazer algumas ações várias vezes. Em uma linguagem puramente funcional você não diz ao computador o que fazer dessa forma mas ao invés disso você diz o que uma coisa é." [1, tradução livre]

Haskell

- Haskell é uma linguagem puramente funcional
  - Chamamos uma linguagem de puramente funcional quando ela não permite efeitos colaterais
  - Como consequência, estruturas de dados em Haskell são imutáveis (não podem ser alteradas)
  - Problemas em Haskell são geralmente resolvidos com recursão
- Haskell é uma linguagem não-estrita
  - O que significa que valores são computados de forma *lazy*, apenas no momento em que são necessários
- Haskell é uma linguagem referencialmente transparente
  - Definições em Haskell respeitam a igualdade matemática: um termo pode ser livremente trocado por sua definição
  - O que só acontece devido à falta de efeitos colaterais

Haskell

- Haskell é uma linguagem puramente funcional
  - Chamamos uma linguagem de puramente funcional quando ela não permite efeitos colaterais
  - Como consequência, estruturas de dados em Haskell são imutáveis (não podem ser alteradas)
  - Problemas em Haskell são geralmente resolvidos com recursão
- Haskell é uma linguagem não-estrita
  - O que significa que valores são computados de forma *lazy*, apenas no momento em que são necessários
- Haskell é uma linguagem referencialmente transparente
  - Definições em Haskell respeitam a igualdade matemática: um termo pode ser livremente trocado por sua definição
  - O que só acontece devido à falta de efeitos colaterais

Haskell

- Haskell é uma linguagem puramente funcional
  - Chamamos uma linguagem de puramente funcional quando ela não permite efeitos colaterais
  - Como consequência, estruturas de dados em Haskell são imutáveis (não podem ser alteradas)
  - Problemas em Haskell são geralmente resolvidos com recursão
- Haskell é uma linguagem não-estrita
  - O que significa que valores são computados de forma *lazy*, apenas no momento em que são necessários
- Haskell é uma linguagem referencialmente transparente
  - Definições em Haskell respeitam a igualdade matemática: um termo pode ser livremente trocado por sua definição
  - O que só acontece devido à falta de efeitos colaterais

Haskell

- Haskell é uma linguagem puramente funcional
  - Chamamos uma linguagem de puramente funcional quando ela não permite efeitos colaterais
  - Como consequência, estruturas de dados em Haskell são imutáveis (não podem ser alteradas)
  - Problemas em Haskell são geralmente resolvidos com recursão
- Haskell é uma linguagem não-estrita
  - O que significa que valores são computados de forma *lazy*, apenas no momento em que são necessários
- Haskell é uma linguagem referencialmente transparente
  - Definições em Haskell respeitam a igualdade matemática: um termo pode ser livremente trocado por sua definição
  - O que só acontece devido à falta de efeitos colaterais

Haskell

- Haskell é uma linguagem puramente funcional
  - Chamamos uma linguagem de puramente funcional quando ela não permite efeitos colaterais
  - Como consequência, estruturas de dados em Haskell são imutáveis (não podem ser alteradas)
  - Problemas em Haskell são geralmente resolvidos com recursão
- Haskell é uma linguagem não-estrita
  - O que significa que valores são computados de forma *lazy*, apenas no momento em que são necessários
- Haskell é uma linguagem referencialmente transparente
  - Definições em Haskell respeitam a igualdade matemática: um termo pode ser livremente trocado por sua definição
  - O que só acontece devido à falta de efeitos colaterais

Haskell

- Haskell é uma linguagem puramente funcional
  - Chamamos uma linguagem de puramente funcional quando ela não permite efeitos colaterais
  - Como consequência, estruturas de dados em Haskell são imutáveis (não podem ser alteradas)
  - Problemas em Haskell são geralmente resolvidos com recursão
- Haskell é uma linguagem não-estrita
  - O que significa que valores são computados de forma *lazy*, apenas no momento em que são necessários
- Haskell é uma linguagem referencialmente transparente
  - Definições em Haskell respeitam a igualdade matemática: um termo pode ser livremente trocado por sua definição
  - O que só acontece devido à falta de efeitos colaterais

Haskell

- Haskell é uma linguagem puramente funcional
  - Chamamos uma linguagem de puramente funcional quando ela não permite efeitos colaterais
  - Como consequência, estruturas de dados em Haskell são imutáveis (não podem ser alteradas)
  - Problemas em Haskell são geralmente resolvidos com recursão
- Haskell é uma linguagem não-estrita
  - O que significa que valores são computados de forma *lazy*, apenas no momento em que são necessários
- Haskell é uma linguagem referencialmente transparente
  - Definições em Haskell respeitam a igualdade matemática: um termo pode ser livremente trocado por sua definição
  - O que só acontece devido à falta de efeitos colaterais

Haskell

- Haskell é uma linguagem puramente funcional
  - Chamamos uma linguagem de puramente funcional quando ela não permite efeitos colaterais
  - Como consequência, estruturas de dados em Haskell são imutáveis (não podem ser alteradas)
  - Problemas em Haskell são geralmente resolvidos com recursão
- Haskell é uma linguagem não-estrita
  - O que significa que valores são computados de forma *lazy*, apenas no momento em que são necessários
- Haskell é uma linguagem referencialmente transparente
  - Definições em Haskell respeitam a igualdade matemática: um termo pode ser livremente trocado por sua definição
  - O que só acontece devido à falta de efeitos colaterais

Haskell

- Haskell é uma linguagem puramente funcional
  - Chamamos uma linguagem de puramente funcional quando ela não permite efeitos colaterais
  - Como consequência, estruturas de dados em Haskell são imutáveis (não podem ser alteradas)
  - Problemas em Haskell são geralmente resolvidos com recursão
- Haskell é uma linguagem não-estrita
  - O que significa que valores são computados de forma *lazy*, apenas no momento em que são necessários
- Haskell é uma linguagem referencialmente transparente
  - Definições em Haskell respeitam a igualdade matemática: um termo pode ser livremente trocado por sua definição
  - O que só acontece devido à falta de efeitos colaterais

Bibliografia

Lipovaca, M.
Learn You a Haskell for Great Good!: A Beginner's Guide, 1st ed.
No Starch Press, San Francisco, CA, USA, 2011.