



Paradigma Funcional

Prof^a. Rachel Reis
rachel@inf.ufpr.br



Paradigma Funcional

- Características de uma linguagem funcional:
 - Uso de funções puras
 - Uso de recursão
 - Avaliação Preguiçosa
 - Cálculo lambda



Paradigma Funcional

- Características de uma linguagem funcional:
 - Uso de **funções puras**
 - Uso de recursão
 - Avaliação Preguiçosa
 - Cálculo lambda



Efeito colateral e estados

- Um efeito colateral ocorre quando uma função altera algum estado global do sistema.
- Exemplo:
 - Alterar uma variável global
 - Ler entrada de dados
 - Imprimir algo na tela



Funções puras

- São funções que não apresentam efeito colateral.
- Ao executar uma função X com a mesma entrada, sempre se obtém a mesma resposta.



Pergunta 1

- Função pura só existe se a linguagem for funcional?
 - Resposta: não.

```
int calcularDobro(int num)
{
    return 2 * num;
}
```



Funções puras e impuras

- A função abaixo é pura ou impura?

```
int i = 0;

int calcularDobroMaisI(int num) {
    i = i + 1;
    return 2 * num + i;
}
```

- Resposta: impura, pois ela depende de um estado que não é definido pelo seu parâmetro.



Exercício 1

- Classifique as seguintes funções em C como pura ou impura:
 - strlen - pura
 - printf - impura
 - getc - impura



Avaliação de Funções

- Função 1: pura ou impura? Impura.

```
void soma_valor(double *soma, int valor)
{
    *soma += valor;
}
```



Avaliação de Funções

- Função 2: pura ou impura? Impura.

```
double calcula_media(int valores[], int n){  
    double soma = 0;  
    int i;  
  
    for(i = 0; i < n; i++){  
        soma_valor(&soma, valores[i]);  
    }  
    return soma/n;  
}
```



Avaliação de Funções

- Funções **impuras** são **virais**!
 - Se uma função X chama uma função Y que é impura, então X é impura.



Pergunta 2

- Se uma função X só chama funções puras, X é sempre pura?

```
int i = 0;

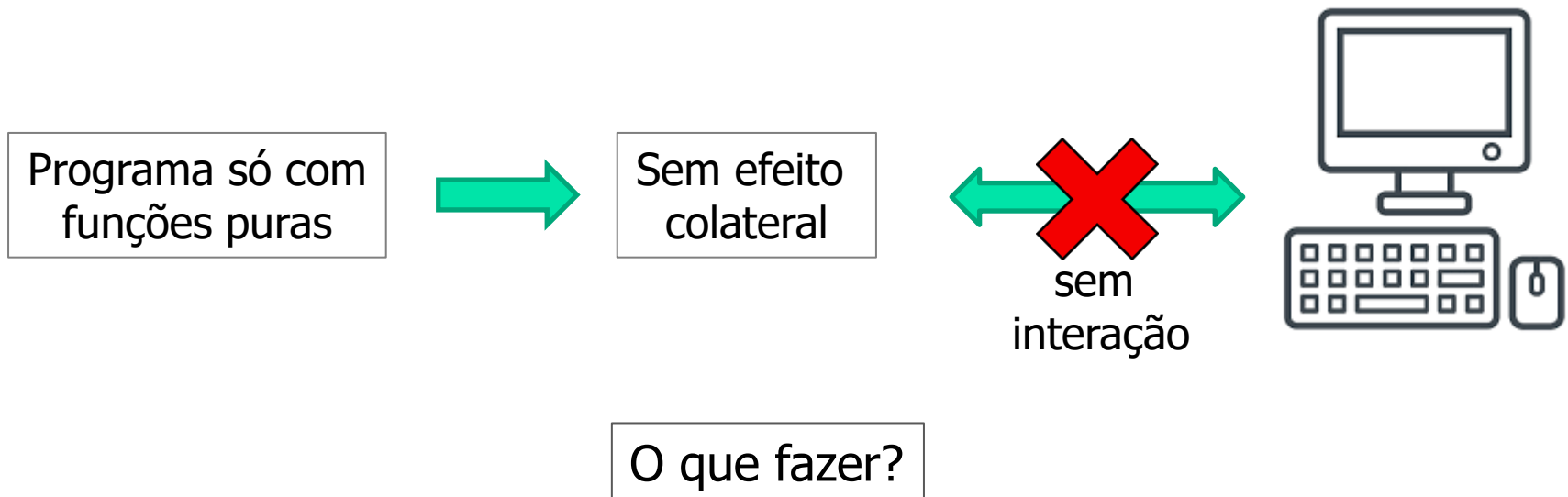
int calcularDobroMaisI(int num) {
    return 2 * num + i;
}
```

- Resposta: não, pois a função X pode depender de algo que não é definido totalmente pelos seus parâmetros.



Pergunta 3

- Um programa que contém apenas funções puras é útil?



- Haskell: deixar as impurezas somente para o ambiente de execução.



Funções puras

- Quais as vantagens de não se ter efeito colateral?
 - Se o resultado de uma expressão pura não for utilizado, ele não precisa ser calculado.
 - O programa como um todo pode ser reorganizado e otimizado.
 - É possível computar expressões em qualquer ordem (ou até em paralelo).



Resumo

Funções Puras



não têm efeito colateral



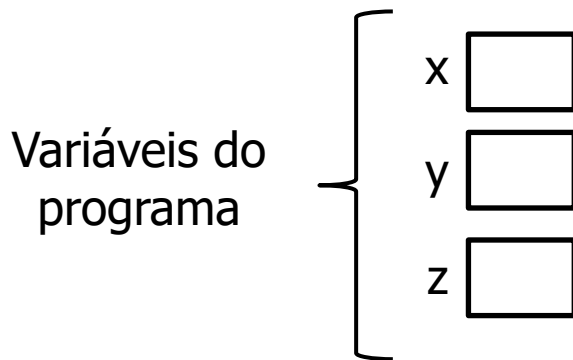
não alteram o estado global do sistema

O que são estados do sistema?



Programação sem bugs

- Os estados do sistema são fontes de muitos problemas, logo a ausência de estados permite evitar muitos erros de implementação.
- O que são **estados do sistema**?

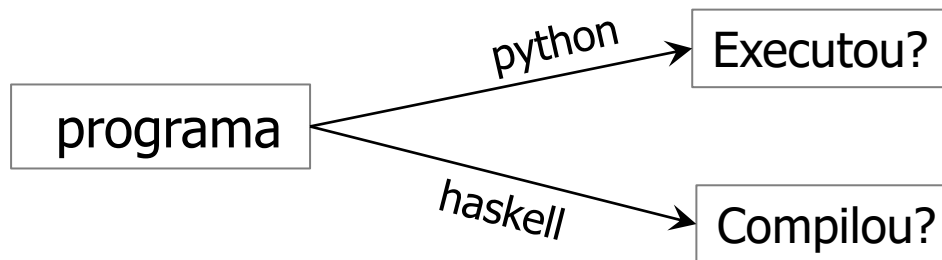


Conteúdo das variáveis em um determinado ponto na execução do programa



Programação sem bugs

- Lema do Haskell: “se compilou, o código está correto!”
- Formas diferentes de se trabalhar com uma linguagem:





Características

- Características de uma linguagem funcional:
 - Uso de funções puras
 - **Uso de recursão**
 - Avaliação Preguiçosa
 - Cálculo lambda



Iteração x Recursão

- Em linguagens funcionais, os laços iterativos são implementados via recursão.
 - Como consequência, tem-se um código mais enxuto e declarativo (mostra o que precisa ser feito e não como).

- C/Java (iterativo)

```
int mdc (int a, int b) {  
    int resto;  
    while (b != 0) {  
        resto = a % b;  
        a = b;  
        b = resto;  
    }  
    return a;  
}
```

- Haskell (recursivo)

```
mdc 0 b = b  
mdc a 0 = a  
mdc a b = mdc b (a `rem` b)
```



Paradigma Funcional

- Características de uma linguagem funcional:
 - Uso de funções puras
 - Uso de recursão
 - **Avaliação Preguiçosa**
 - Cálculo lambda



Avaliação Preguiçosa

- Avaliação preguiçosa (*lazy evaluation*) também conhecida com avaliação sob demanda.
- Quando uma expressão é gerada, ela gera uma promessa de execução (*thunk*).
- Se em algum momento o valor gerado pela expressão for necessário, o *thunk* é avaliado.



Exemplo em C - Avaliação Estrita

- Avaliação estrita: oposto da avaliação preguiçosa, ou seja, os valores/expressões são sempre avaliados.

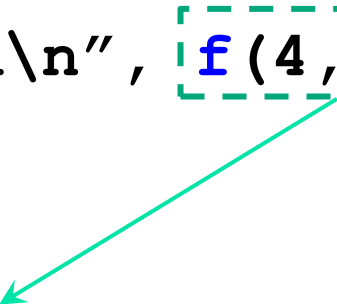
```
int main() {  
    int x = 2;  
    printf("%d\n", f(x * x, 4 * x + 3));  
    return 0;  
}  
  
int f(int x, int y) {  
    return 2 * x;  
}
```



Exemplo em C - Avaliação Estrita

- Avaliação estrita: oposto da avaliação preguiçosa, ou seja, os valores/expressões são sempre avaliados.

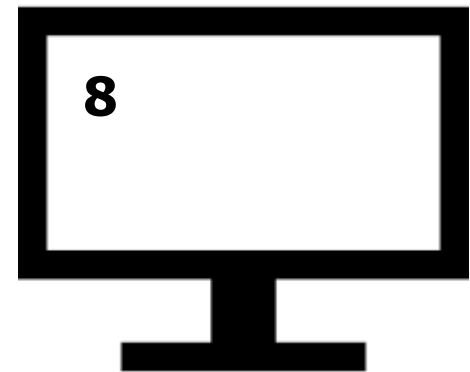
```
int main() {  
    int x = 2;  
    printf("%d\n", f(4, 11));  
    return 0;  
}  
  
int f(int x, int y) {  
    return 2 * x;  
}
```



Exemplo em C - Avaliação Estrita

- Avaliação estrita: oposto da avaliação preguiçosa, ou seja, os valores/expressões são sempre avaliados.

```
int main() {  
    int x = 2;  
    printf("%d\n", 8);  
    return 0;  
}  
  
int f(int x, int y) {  
    return 2 * x;  
}
```





Avaliação Preguiçosa

- Exemplo em Haskell.

```
f x y = 2 * x
```

```
main = do
```

```
    let z = 2
```

```
    print (f (z * z) (4 * z + 3))
```

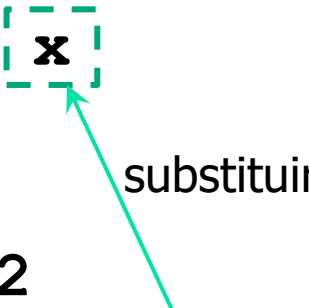


Avaliação Preguiçosa

- Exemplo em Haskell.

```
f x y = 2 * x  
  
main = do  
    let z = 2  
    print (f (z * z) (4 * z + 3))
```

substituir





Avaliação Preguiçosa

- Exemplo em Haskell.

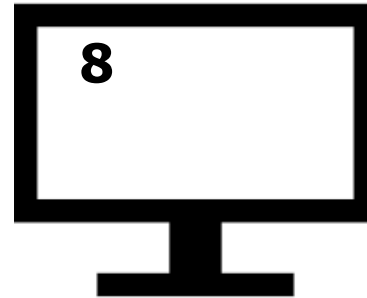
```
f x y = 2 * x  
  
main = do  
    let z = 2  
    print (2 * (z * z))
```

Avaliação Preguiçosa

- Exemplo em Haskell.

```
f x y = 2 * x
```

```
main = do  
    let z = 2  
    print (8)
```



- A segunda parte da expressão nunca foi avaliada!

```
print (f (z * z) (4 * z + 3))
```



Avaliação Preguiçosa

- Isso permite a criação de listas infinitas.

```
[2 * i | i <- [0..]]
```



Paradigma Funcional

- Características de uma linguagem funcional:
 - Uso de funções puras
 - Uso de recursão
 - Avaliação Preguiçosa
 - **Cálculo lambda**



Cálculo Lambda

- Considerado uma das características fundamentais das linguagens funcionais.
- É a base teórica para a programação funcional e possui influência direta no *design* e na semântica dessas linguagens.



Cálculo Lambda

- Nas linguagens de programação vistas até agora temos:
 - Atribuições
 - Condicionais
 - Laços
 - Funções
 - Recursão
 - Ponteiros
 - Classes e objetos



Cálculo Lambda

- O cálculo lambda (λ) descreve a computação utilizando **apenas funções**.
 - ~~Atribuições~~
 - ~~Condicionais~~
 - ~~Laços~~
 - **Funções**
 - ~~Recursão~~
 - ~~Ponteiros~~
 - ~~Classes e objetos~~



Cálculo Lambda

- Conjunto de regras e símbolos que nos ajudam a fazer “coisas” com funções.
- Exemplos:
 - Criar funções sem dar um nome específico (funções anônimas)
 - Criar funções que retornam outras funções (funções de alta ordem)
 - Transformar uma função que recebe vários argumentos em uma sequência de funções que recebe um único argumento (*currying*)



Linguagem do Cálculo Lambda

- Formada por três elementos:
 - Variáveis
 - Definição de funções
 - Aplicação de funções



Linguagem do Cálculo Lambda

- Formada por três elementos:
 - Variáveis
 - Definição de funções
 - Aplicação de funções
- Na sintaxe original, a letra grega minúscula lambda (λ) é usada para definir funções.



Exemplo 1

- Vamos criar uma função lambda que soma dois números.

$$(\lambda x. \lambda y. x + y)$$

Lê-se: a função lambda recebe dois argumentos e retorna o resultado da soma de x e y.

No cálculo lambda, uma função é definida utilizando a notação λ (lambda), seguido pelos argumentos e o corpo da função.



Exemplo 1

- Vamos criar uma função lambda que soma dois números.

$$(\lambda x. \lambda y. x + y)$$

Lê-se: a função lambda recebe dois argumentos e retorna o resultado da soma de x e y.

- Aplicando a função lambda aos valores 3 e 5

$$((\lambda x. \lambda y. x + y) 3 5)$$

→ Resultado: $3 + 5 = 8$



Exemplo 2

- Vamos criar uma função lambda que multiplica um número por 2.

$$(\lambda x. x * 2)$$

Lê-se: a função lambda recebe um argumento e multiplica por 2.

No cálculo lambda, uma função é definida utilizando a notação λ (lambda), seguido pelos argumentos e o corpo da função.



Exemplo 2

- Vamos criar uma função lambda que multiplica um número por 2.

$$(\lambda x. x * 2)$$

Lê-se: a função lambda recebe um argumento e multiplica por 2.

- Aplicando a função lambda ao valor 4

$$((\lambda x. x * 2) 4)$$

→ Resultado: $4 * 2 = 8$



Linguagem do cálculo lambda

- Sintaxe do Haskell troca:

- λ por \backslash
- $.$ por \rightarrow

- Logo,

$(\lambda x. \lambda y. x + y)$

$(\backslash x \rightarrow \backslash y \rightarrow x + y)$

$(\lambda x. x^2)$

$(\backslash x \rightarrow x^2)$



Paradigma Funcional

- Muitas linguagens de programação estão incorporando elementos do paradigma funcional por conta dos seus benefícios.

Exemplo Java

```
interface Calculadora {  
    int calcularQuadrado(int numero);  
}  
  
public class Principal {  
    public static void main(String[] args) {  
        /* Definindo uma expressão lambda para calcular o  
           quadrado de um número */  
        Calculadora calculadora = (valor) -> valor * valor; Função anônima  
  
        /* Usando a expressão lambda para calcular o  
           quadrado do número 5*/  
        int resultado = calculadora.calcularQuadrado(5);  
        System.out.println("O quadrado de 5 é: " + resultado);  
    }  
}
```



Por que Haskell?

- Linguagem puramente funcional (não é multi-paradigma)
- Aceita somente funções puras (como tratar entrada e saída de dados?)
- Declarativa (especifica o que o programa faz)
- Uso de recursão
- Avaliação preguiçosa
- Cálculo lambda
- ...



Referências

- Oliveira, A. G. de (2017). Haskell – Uma introdução à programação funcional. Casa do Código.
- Curso de paradigmas de programação (Haskell) da Universidade Federal do ABC (UFABC). Disponível em <<https://www.youtube.com/watch?v=eTisiy5FB7k&list=PLYltvall0TqJ25sVTLcMhxsE0Hci58mpQ>>. Último acesso em 23/01/2023.