Programação Funcional

(COMP0393)

Leila M. A. Silva

dcomp
departamento de computação

Projeto e Escrita de Programas

(COMP0393)

Aula 5

dcomp
departamento de computação

Projeto e Implementação

- Projetar
  - Planejamento prévio à implementação
  - Arquitetar uma solução
  - Tudo o que fazemos ANTES de implementar
- Implementar
  - Escrever um programa

dcomp departamento de computação

Entendendo o Problema

- Entender o que se deseja resolver é o primeiro passo
- A linguagem informal pode não deixar totalmente claro o que se deseja.
- Ex: Escreva uma função para retornar o elemento do meio de três números dados.
  - Se os números forem 2,4,3 é fácil! A resposta seria 3.
  - Mas e para 2,4,2?
    - 2? Porque 2,2,4
    - Não existe elemento do meio.

Entendendo o Problema

- Mesmo para problemas simples podem existir detalhes a serem pensados
- Não há resposta correta – programador e usuário devem definir claramente o que desejam em todos os casos
- Exemplos são facilitadores para exemplificar as diversas situações
- Quanto mais cedo descobrirmos falhas no entendimento do problema, menos esforço de retrabalho (correções de erros)

Entendendo o Problema

- É importante definir os tipos de sua função cedo, assim ficará claro quais e quantos argumentos terá e qual o tipo do seu resultado.
- Use nomes adequados para sua função para que o código fique legível
- Ex: `valorDoMeio :: Int -> Int -> Int -> Int`

O que posso usar na minha solução?

- Funções que eu já implementei
- Funções pré-definidas no Prelude ou bibliotecas
- Pensamento *bottom-up*

O que posso usar na minha solução?

- Ex: Calcular o maior de três números
  - Já definimos a função `maxi`, podemos usá-la de modelo
  - Na realidade o Prelude tem a função `max` pré-definida

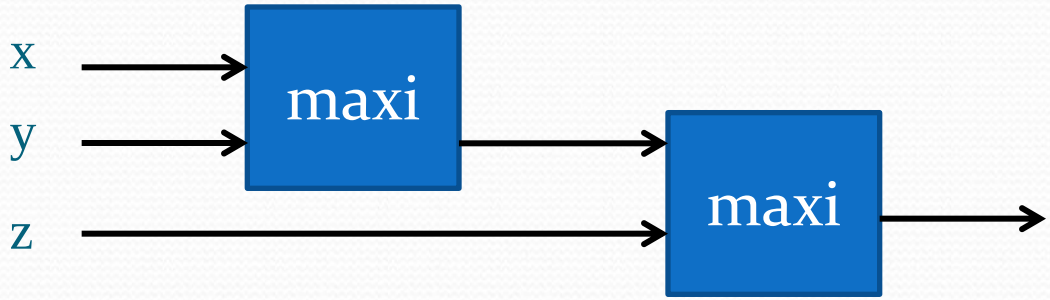
```
maxi :: Int -> Int -> Int
maxi x y
     | x >= y = x
     | otherwise = y

maxiTres :: Int -> Int -> Int -> Int
maxiTres x y z
| x >= y && x >= z = x
| y >= x && y >= z = y
| otherwise = z
```

O que posso usar na minha solução?

- Mas, será que não poderíamos usar `maxi` na definição de `maxiTres`?

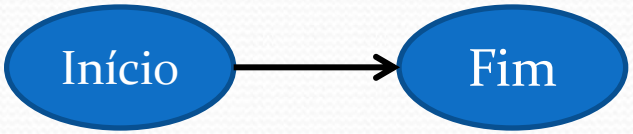
```
maxiTres :: Int -> Int -> Int -> Int
maxiTres x y z = maxi (maxi x y) z
```



- Bem mais compacto e elegante, não???

Decomposição de Problemas

- Pensamento *top-down*
- Será que posso quebrar meu problema em problemas menores e mais fáceis de serem resolvidos? **Divisão**
- Como posso combinar soluções dos problemas menores para obter a solução do problema maior? **Conquista**
- Decomposição usa abstração: Será que eu tenho uma função que eu preciso?



Decomposição de Problemas

- Ex: Calcular o elemento do meio de três números

```
valorDoMeio :: Int -> Int -> Int -> Int
valorDoMeio x y z
    | condicao para x = x
    | condicao para y = y
    | otherwise = z
```

- Com o esquema pronto, preciso agora pensar quais condições vou colocar para `x` e para `y`
- Será que posso criar uma função para os dois casos?
- Observe que o elemento do meio, seja ele quem for, obedece à seguinte propriedade: meio >= esquerdo e meio <= direito

dcomp departamento de computação

Decomposição de Problemas

```
valorDoMeio :: Int -> Int -> Int -> Int
valorDoMeio x y z
    | entre y x z = x
    | entre x y z = y
    | otherwise = z
    where entre a b c = (a <= b) && (b <= c)
                      || (a >= b) && (b >= c)
```

- Desta forma usamos a função `entre` como auxiliar na solução da função `valorDoMeio`

Exercício de Fixação

- Usando `maxi` elabore uma função para achar o maior valor de quatro números inteiros.
- Usando `maxi` e `maxiTres` elabore uma função para achar o maior valor de cinco números inteiros.

Exercício de Fixação

- Usando `maxi` elabore uma função para achar o maior valor de quatro números inteiros.

```
maxiQuatro :: Int -> Int -> Int -> Int-> Int
maxiQuatro x y z u= maxi (maxi x y) (maxi z u)
```

- Usando `maxi` e `maxiTres` elabore uma função para achar o maior valor de cinco números inteiros.

```
maxiCinco :: Int -> Int -> Int -> Int -> Int -> Int
maxiTres x y z u v = maxiTres (maxi x y) (maxi z u) v
```

dcomp departamento de computação

Exercício de Fixação

- Dado dois inteiros elabore uma função para retornar o máximo entre eles e o número de vezes que ele ocorre

Exercício de Fixação

- Dado dois inteiros elabore uma função para retornar o máximo entre eles e o número de vezes que ele ocorre

```
maxOcorre :: Int -> Int -> (Int,Int)
maxOcorre x y
   | x == y = (x, 2)
   | otherwise = ((max x y), 1)
```

dcomp departamento de computação

Decomposição de Problemas

- Problema: Considere o problema de atualizar o *i*-ésimo valor de uma lista de inteiros, com o seu dobro.
  - Ex: `[1,2,3,4,5]` e desejo atualizar o terceiro valor, ficando `[1,2,6,4,5]`
- Que operadores e funções posso usar?
  - O operador !! permite acessar o elemento, mas como ele começa a contar de zero, preciso me lembrar que o *i*-ésimo está na posição *i*-1!
  - Mas o operador não permite que eu atualize...
  - Vamos estudar outras formas de fazer isto mais adiante, mas usando as funções que já temos, vamos tentar decompor o problema em problemas menores

dcomp departamento de computação

Decomposição de Problemas

- Problema: Considere o problema de atualizar o *i*-ésimo valor de uma lista de inteiros, com o seu dobro.
  - Ex: `[1,2,3,4,5]` e desejo atualizar o terceiro valor, ficando `[1,2,6,4,5]`
- Analisando o problema, observe que temos:
  - Duas partes da lista que não sofrerão alterações `[1,2,3,4,5]`
  - Um elemento a ser alterado entre as duas partes `[1,2,6,4,5]`
- Como coleto as partes das listas que não se alteram?
  - Usando `take` e `drop`!
- Como modifico o *i*-ésimo valor?
  - Usando `dobro`!
- Como gero o resultado final?
  - Usando ++!

dcomp departamento de computação

Decomposição de Problemas

- Problema: Considere o problema de atualizar o *i*-ésimo valor de uma lista de inteiros, com o seu dobro.
  - Ex: `[1,2,3,4,5]` e desejo atualizar o terceiro valor, ficando `[1,2,6,4,5]`
- Podemos tentar escrever

```
atualizaValor :: Int -> [Int] -> [Int]
atualizaValor  i xs = parteUm ++ [valorAtual] ++ parteDois
   where parteUm = take (i-1) xs
         valorAtual = dobro (xs!!(i-1))
         parteDois = drop i xs
```

Decomposição de Problemas

- Será que a função está mesmo correta? Será que todos os casos estão contemplados?
- E se a lista for vazia?
- E se o valor de *i* informado for inválido? Quais valores de *i* são inválidos para este problema?
- É preciso também pensar nos casos de insucesso!!! E decidir como vai tratá-los!

```
atualizaValor2 :: Int -> [Int] -> [Int]
atualizaValor2 _ [] = []
atualizaValor2  i xs
  | indiceOK  = parteUm ++ [valorAtual] ++ parteDois
  | otherwise = error " valor de i invalido"
  where indiceOK = (i > 0) && (i <= length xs)
        parteUm = take (i-1) xs
        valorAtual = dobro (xs!!(i-1))
        parteDois = drop i xs
```

dcomp departamento de computação

Decomposição de Problemas

- Boa prática na análise do problema
  - Entenda o problema explorando com vários exemplos
  - Estabeleça a declaração da função que deseja
  - Identifique o caso geral de sucesso
  - Identifique os casos de insucesso
  - Resolva o caso geral de sucesso
  - Introduza os casos de insucesso
  - Analise a solução proposta. O que aprendi? A solução pode ser melhorada (refatoração)? Há outras soluções? ...

Decomposição de Problemas

- No exemplo anterior, sabíamos a posição da lista que precisava ser atualizada e o operador !! foi usado para pegar o elemento desta posição. Mas e se eu não souber a posição?
- Dado um elemento `(x,y)` e uma lista de `(Int,Int)` escreva uma função para atualizar o elemento dado com `(x, x+y)`. Os demais elementos permanecem sem alteração. Considere que o par é único, ou seja, não existem dois pares na lista com os mesmos valores.
- Ex: Atualizar o (10,9) de [(1,2), (4,5), (10,9)] retornaria

  [(1,2), (4,5), (10,19)]

dcomp departamento de computação

Decomposição de Problemas

- Dado um elemento `(x,y)` e uma lista de `(Int,Int)` escreva uma função para atualizar o elemento dado com `(x, x+y)`. Os demais elementos permanecem sem alteração. Considere que o par é único, ou seja, não existem dois pares na lista com os mesmos valores.
- Posso usar a questão anterior como modelo e separar o problema nas partes que se alteram e que não se alteram.
- Mas como saber qual parte da lista não se altera se não sei onde está o elemento?
- Preciso então localizar onde ele está!!!!

dcomp departamento de computação

Decomposição de Problemas

- Dado um elemento `(x,y)` e uma lista de `(Int,Int)` escreva uma função para atualizar o elemento dado com `(x, x+y)`. Os demais elementos permanecem sem alteração. Considere que o par é único, ou seja, não existem dois pares na lista com os mesmos valores.
- Como localizo a posição do elemento?
  - Preciso ter uma forma de registrar a posição da lista...

```
indexaLista :: [(Int,Int)] -> [(Int, (Int,Int))]
indexaLista xs = zip ls xs
   where ls = [1..(length xs)]
```

Decomposição de Problemas

- Como localizo a posição do elemento?
  - Agora que a lista está indexada posso coletar esta posição, se o elemento existir. Caso contrário a função retorna zero, sinalizando que a coleta falhou

```
encontraPosElem :: (Int,Int)->[(Int, (Int,Int))]->Int
encontraPosElem (x,y) xys
  | ps == [] = 0
  | otherwise = head ps
  where ps =  [p| (p, (u,v))<-xys, (u==x && v==y)]
```

Decomposição de Problemas

- Posso então construir uma função similar a `atualizaValor` para o caso de sucesso

```
atualizaElem :: (Int,Int) ->[(Int,Int)] -> [(Int,Int)]
atualizaElem (x,y) xys = parteUm ++ [valorAtual] ++ parteDois
   where pos = encontraPosElem (x,y) (indexaLista xys)
         parteUm = take (pos-1) xys
         valorAtual = atualizaPar (xys!!(pos-1))
         parteDois = drop pos xys
         atualizaPar :: (Int, Int) -> (Int, Int)
         atualizaPar (u,v) = (u, u+v)
```

Decomposição de Problemas

- Agora posso pensar nos casos de insucesso.
- Como já tratamos em `encontraPosElem` o caso do elemento não estar na lista, aqui precisamos somente decidir o que fazer quando `pos` for zero e tratar o caso em que o elemento não ocorre na lista

```
atualizaElem :: (Int,Int) ->[(Int,Int)] -> [(Int,Int)]
atualizaElem (x,y) xys
  | indiceOk = parteUm ++ [valorAtual] ++ parteDois
  | otherwise = (x,x+y): xys
   where pos = encontraPosElem (x,y) (indexaLista xys)
         indiceOK = (pos /= 0)
         parteUm = take (pos-1) xys
         valorAtual = atualizaPar (xys!!(pos-1))
         parteDois = drop pos xys
         atualizaPar :: (Int, Int) -> (Int, Int)
         atualizaPar (u,v) = (u, u+v)
```

Exercícios Recomendados

- Dados três inteiros determine quantos deles são iguais.
- Usando esta função resolva agora o mesmo problema para quatro inteiros.
- Escreva uma função para atualizar o segundo elemento do *i*-ésimo par de uma lista de pares, com o produto do primeiro valor do par pelo segundo valor.
  - Ex: Atualizar o segundo par de [(1,2), (4,5), (10,9)] retornaria

    [(1,2), (4,20), (10,9)]
- Escreva uma função para atualizar uma tupla *(x,y,z)* de uma lista de tuplas de três elementos inteiros, com uma tupla da forma *(x, x+y, x+y+z)*.
  - Ex: Atualizar o (4,5,1) de [(1,2,0), (4,5,1), (10,9,2)] retornaria

    [(1,2,0), (4,9,10), (10,9,2)]
- Tente pensar uma solução para a questão anterior que não faz uso da posição do elemento na lista para atualizar a tupla *(x,y,z)*.

dcomp departamento de computação

Indução Matemática

- A técnica de indução matemática é extremamente útil para reduzir problemas maiores em problemas menores
- Indução Fraca => recursão primitiva
- Indução Forte => recursão geral
- Vamos primeiro explorar o princípio da indução matemática e depois aprender a construir problemas com ele.
- Mas antes disso... Depois que implemento meu programa, como faço para testá-lo?

dcomp
departamento de computação

Testes

- O programa compilou!!! Mas, ele está certo???
- Provas – veremos mais tarde
- Testes
  - Caixa preta – escolhe casos de teste baseados na descrição do problema, independente do código implementado
  - Caixa branca – escolhe casos de teste que exercitem todos os caminhos da função implementada

dcomp departamento de computação

Testes

- Caixa preta – `maxiTres`
  - Três valores iguais. Ex: 4 4 4
  - Três valores diferentes: Ex: 6 4 1
  - Dois dos valores iguais:
    - Dois valores iguais ao máximo, um diferente. Ex: 6 6 1
    - Um valor máximo, dois outros iguais. Ex: 6 1 1

Testes

- Caixa Branca
  - Exercitar todos as guardas e ramos de condicionais
  - Explorar situações limites
  - Explorar o caso base e geral em funções recursivas (ainda iremos ver)

Exercício de Fixação

- Pense quais seriam os casos de teste para uma função que dado três inteiros determina se são todos iguais.

Exercício de Fixação

- Pense quais seriam os casos de teste para uma função que dado três inteiros determina se são todos iguais.

- Casos:
  - Três iguais
  - Dois iguais – que na realidade são três possibilidades (*x* e *y*, *x* e *z*, e *y* e *z* iguais)
  - Todos diferentes