A Cifra de César

Programação Funcional em Haskell

Prof. Rodrigo Ribeiro

Cifra de César — (I)

- Algoritmo de criptografia utilizado por César para se comunicar com seus generais.
- Idéia do algoritmo
  - Substituir cada caractere c por c + n, em que n é um número que representa um deslocamento.

Cifra de César — (II)

- Projeto usando programação funcional
  - Como transformar a entrada do algoritmo na saída dele?
  - Quais os tipos da entrada e saída?
- Entradas para o algoritmo:
  - Texto a ser encriptado
  - Valor do deslocamento.
  - Tipos: String e Int
- Saída: Texto encriptado
  - String

Cifra de César — (III)

- Como transformar a entrada na saída?
  - Strings = [Char]...
  - Logo, basta percorrer a lista e fazer o deslocamento de cada caractere
- Como realizar o deslocamento de caracteres?
  - Ex: Deslocar 'a' de 2 deve produzir 'c'...

Cifra de César — (IV)

- Algumas configurações necessárias...
  - Módulo Data.Char possui diversas operações sobre o tipo Char.

```
module Main where

import Data.Char (chr,ord)
```

Cifra de César — (V)

- Deslocando um caractere

```
shift' :: Int -> Char -> Char
shift' n c = chr ((ord c) + n)
```

Cifra de César — (VI)

- Criptografando!

```
encode' :: Int -> String -> String
encode' n [] = []
encode' n (c:cs) = shift' n c : encode' n cs
```

Cifra de César — (VII)

- Descriptografando...

```
decode' :: Int -> String -> String
decode' n [] = []
decode' n (c:cs) = shift' (-n) c : decode' n cs
```

- Sendo um pouco mais esperto...

```
decode' :: Int -> String -> String
decode' n s = encode' (-n) s
```

Cifra de César — (VIII)

- Mas isso está correto?
  - Isso é, realmente a encriptar e depois descriptografar vai produzir a mensagem original?

```
test :: (Int -> String -> String) -> -- encriptar
        (Int -> String -> String) -> -- descriptografar
        Int ->                       -- deslocamento
        String ->                    -- texto original
        Bool                         -- resultado
test enc dec n s = dec n (enc n s) == s
```

Cifra de César — (IX)

- Alguns comentários sobre essa solução:
  - Uso de recursão diretamente não é aconselhável.
  - Porquê: em programação funcional padrões de recursão são encapsulados por funções de ordem superior.
- Qual o padrão de recursão utilizado pelo algoritmo da cifra de César?

Cifra de César — (X)

- Notem que o padrão de recursão utilizado é:
  - Aplicar uma função a cada um dos elementos de uma lista.
- Esse padrão de recursão é expresso pela função map:

```
map :: (a -> b) -> [a] -> [b]
map f [] = []
map f (x:xs) = f x : map f xs
```

Cifra de César — (XI)

- Uma breve pausa... Lembram-se da função foldr?

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f v [] = v
foldr f v (x:xs) = f x (foldr f v xs)
```

- É possível definir map usando foldr?

Cifra de César — (XII)

- Definindo map utilizando foldr:

```
map :: (a -> b) -> [a] -> [b]
map f = foldr step []
        where
           step x ac = f x : ac
```

Cifra de César — (XIII)

- Solução apresentada usa uma definição local meramente para aplicar a função f, existe outra maneira de fazer isso?
  - Pode ser cansativo ficar digitando 'where'...
- Sim! Basta usar uma função anônima!
  - Nesse exemplo, temos uma função anônima com dois parâmetros, a saber: x e ac.

```
map :: (a -> b) -> [a] -> [b]
map f = foldr (\x ac -> f x : ac) []
```

Cifra de César — (XIV)

- Voltando a programação normal...
  - Agora, como definir encode e decode utilizando map?

```
encode'' :: Int -> String -> String
encode'' n = map (\c -> shift' n c)

decode'' :: Int -> String -> String
decode'' n = encode'' (-n)
```

Cifra de César — (XV)

- Mas, o algoritmo está correto?

```
*Main> test encode'' decode'' 2 "abc"
True
```

- Parece que sim...
  - Há espaços para melhorias nesse programa?

Cifra de César — (XVI)

- Evidentemente ainda há espaço para melhorias nesse programa!
- Para isso vamos usar a composição de funções para definir a função de deslocamento.
- Eis a definição da composição:

```
(.) :: (b -> c) -> (a -> b) -> (a -> c)
(f . g) x = f (g x)
```

Cifra de César — (XVII)

- Importância de se usar composição de funções.
  - Eficiência do código produzido devido a otimizações presentes no GHC.
  - Definir funções por composição é normalmente conhecido como point-free style.

Cifra de César — (XVIII)

- Versão final:

```
encode :: Int -> String -> String
encode n = map (chr . (+ n) . ord)

decode :: Int -> String -> String
decode n = encode (-n)
```

Cifra de César — (XIX)

- Questão: Como definir map em termos de foldr utilizando composição de funções ao invés de uma função anônima ou uma definição local?

Cifra de César — (XX)

- Solução

```
map' :: (a -> b) -> [a] -> [b]
map' f = foldr ((:) . f) []
```

- Um pequeno teste...

```
*Main> map' (*2) [1,2,3]
[2,4,6]
```

Cifra de César — (XXI)

- Próximas tarefas:
  - Estudar o capítulo 6 do livro Learn you a Haskell for the great good
  - Estudar os capítulos 1,2,3 e 4 do livro Real World Haskell
- Implementar a Cifra de Virgenère em Haskell
  - Marcaremos um dos próximos encontros para que vocês apresentem suas soluções.
- Recomendações para a apresentação:
  - Utilize o pandoc, pois isso irá permitir vocês fazerem os slides usando próprio programa construído por vocês.
  - Procure utilizar o maior número possível de recursos de Haskell aprendidos na solução.
  - Tempo de apresentação: 10 min.

Cifra de César — (XXI)

- Materiais do grupo de estudos de Haskell
  - Estão disponíveis no github: www.github.com
  - Criem uma conta e instalem o software git.
  - Me adicionem (usuário: rodrigogribeiro)
  - Obtenham o material utilizando o git.
- A implementação feita por vocês deve ficar disponível no github.
- Dúvidas sobre como usar o pandoc, git ou dúvidas / dificuldades na implementação do algoritmo fiquem a vontade para me procurar pessoalmente ou por e-mail.