Relatório Projecto LI I

Pedro Faria Helena Alves André Santos

8 de Janeiro de 2011

Conteúdo

1 Introdução

No âmbito da cadeira Laboratórios de Informática I, perante o problema apresentado iremos neste trabalho desenvolver em Haskell um programa que analisa uma lista de inscrições originada por um inquérito online e irá dar resposta a 4 tarefas apresentadas.
De inicio para dar resposta á Tarefa 1 será feita uma limpeza á lista de inscrições apresentada e serão separadas as inscrições referentes a diferentes tipos de inscrições (Alunos, Alunos Externos e Empresas), para além disso serão separadas as inscrições mal preenchidas e serão ignoradas as repetidas e as sem os campos básicos preenchidos.
De seguida (Tarefa 2) o programa irá gerar crachás devidamente preenchidos para cada inscrição válida.
Já no que se refere á Tarefa 3 e 4 o programa irá gerar estatística e análises multi-dimensionais referentes ás inscrições através de gráficos e tabelas que serão apresentados em formato .pdf.

2 Tarefa 1

Após o conhecimento do código dado, começamos a resolver a tarefa 1 em diversas etapas:

2.1 Limpar a lista inicial

Aqui definimos funções que limpam e organizam o código inicial. Começamos por retirar elementos desnecessários, como " \ ", " [" e "] ", através do filter.

```
ltexto1 :: String -> String
ltexto1 k = filter q k
where q k = k/='[' && k/=']' && k/='\"'
```

De seguida, limpamos parte do código referente aos motivos das incrições, dos vários participantes, onde substituímos os acentos pelo código ascii correspondente, como no exmeplo:

```
razoes :: [String] -> [String]
razoes [] = []
razoes (x:xs) | x=="'Procura projecto/disserta\\xc3\\xa7\\xc3\\xa3o de mestrado'"
                = "Procura projecto/disserta\231\227o de mestrado":razoes xs
          (...)
          | otherwise = x:razoes xs
```

Depois, usamos a função lines já pré-definida no Prelude. Com esta função separamos a string inicial pelos \n, obtendo assim uma lista de strings em que cada string será uma inscrição.

Após o uso da função lines, separamos os vários campos das inscrições por vírgulas, obtendo uma lista de listas de strings em que cada string é um campo de inscrição. Temos, assim, uma lista das inscrições com os campos devidamente separados.

```
svirgulas :: String -> [String]
```

2.2 Separar os vários campos de preenchimento

Nesta etapa trabalhamos os campos, começando por criar funções que dêm como resultado cada um desses campos. Estas funções serviram para, no futuro, simplificar o código e simplificar a leitura deste.

```
mail :: [String] -> String
mail x = (last (take 2 x))
```

De notar, que na função relativa ao número de aluno retiramos a letra(s) antecedentes.

`Ex: a54320 -> 54320`

Relativamente, aos campos sobre o jantar da conferência e os almoços no restaurante panorâmico substituímos o ‘‘0’’ e o ‘‘1’’ por ‘‘Nao vai ao jantar’’ e ‘‘Vai ao jantar’’, respectivamente.

`Ex: "0" -> Não vai ao jantar`

No campo das razões da inscrição criamos uma função `pqinsc` que junta as strings correspondentes à resposta de cada participante numa só string.

2.3 Verificação das inscrições

Nesta etapa, definimos uma função que verifica se o nome e o e-mail estão preenchidos, como por exemplo:

```
vnomes :: [[String]] -> [[String]]
vnomes [] = []
vnomes (x:xs) = if nome x =="" then vnomes xs else x: vnomes xs
```

De notar que se algum destes campos não estiver preenchido a inscrição é ignorada, pois são campos de preenchimento obrigatório e essencial.

Também criamos a função `verificacao`, que verifica se as inscrições estão bem preenchidas. Caso estejam e se corresponderem a inscrições de alunos, dá como resultado só os campos que estes devem preeencher. Isto também se aplica às inscrições de universitários e empresas.

```
verificacao :: [[String]] -> [[String]]
verificacao [] = []
verificacao (x:xs) | curso x /="" && numero x /= "" && curso1 x ==""
&& empresa x =="" = [nome x , mail x , curso x , numero x , "Universidade do
Minho" , jantar x , almoco x , pqinsc x , outros x , dat x] : verificacao xs
  (...)
   | otherwise = verificacao xs
```

E, uma função que verifica se existem inscrições repetidas.

```
repetidas1 :: [String] -> [[String]] -> Bool

repetidas :: [[String]] -> [[String]]
repetidas [] = []
repetidas (x:xs)|repetidas1 x xs = repetidas xs
        |otherwise = (x:(repetidas xs))
```

De notar que a inscrição que será ignorada será a primeira a ser preenchida, pois entendemos que se preencheu novamente foi para corrigir a primeira inscrição.

2.4 Criar funções

Aqui criamos funções que separam os diferentes tipos de inscrições: alunos, universitários, empresas. De notar que o resultado será por exemplo no caso da função alunos, uma lista de alunos (Inscrições Válidas) em que cada inscrição de aluno tem apenas o que deveria ser preenchido, ignorando assim campos relativos a outro tipo de inscrição como por exemplo Empresa.

2.5 Criar ficheiro com inscrições

Após a verificação das inscrições, decidimos criar um ficheiro com todas as inscrições, válidas e inválidas. Para isso, começamos por definir a função `alun` que trasforma uma lista de inscrições numa String novamente, separando os campos por ","e as inscrições por \n que no ficheiro .txt será um paragrafo.
De seguida, criamos a função que dá como resultado as inscrições inválidas:

```
mal1 :: [[String]] -> [[String]]
mal1 [] = []
mal1 (x:xs) | curso x /="" && numero x /= "" && curso1 x ==""
              && empresa x =="" = mal1 xs
         (...)
          | otherwise = x: mal1 xs
```

De notar que as inscrições com o nome/mail não preenchidos e as repetidas não são consideradas inválidas, mas serão ignoradas.
E, por fim, definimos a função que cria o ficheiro com as inscrições:

```
inscvalidas1 :: String -> String
inscvalidas1 [] = []
inscvalidas1 k = " Inscrições Válidas \n \n Alunos da Universidade \n \n "
                 ++ alun (alunos k) ++ "\n \n Alunos Externos \n \n"
                 ++ alun (alunosext k) ++ "\n \n Empresas \n \n"
                 ++ alun (empresas k) ++ "\n \n Mal Preenchidas \n \n"
                 ++ alun (mal k)

inscvalidas :: [Char] -> IO ()
inscvalidas k = do writeFile "Inscricoes.txt" (inscvalidas1 k)
```

3 Tarefa 2

Na tarefa 2, começamos por criar um ficheiro LaTeXem que o pdf resultante será composto por folhas A4. As margens destas foram reduzidas de forma a que fosse possível colocar 10 crachás numa só página. Usando o seguinte pacote:

```
\usepackage[a4paper,left=1.25cm,right=1.5cm,top=1cm]{geometry}
```

Para colocar os crachás par a par decidimos escrever em duas colunas:

```
\documentclass[twocolumn]{article}
```

Para além disso criamos uma imagem do tamanho de uma pagina a4 já com 10 crachás e definimos essa imagem como background do documento usando o codigo fornecido no enunciado para tal efeito.

Também usamos o pacote minipage para colocarmos vários crachás na mesma página, sendo a largura de cada crachá de 89m.:

```
\begin{minipage}{89mm}
 CÓDIGO DO CRACHÁ
\end{minipage}
```

Depois de concluído o código, editámo-lo de maneira a que este possa ser lido e dado como output da nossa função. Para isso, fizemos algumas alterações:

- Retiramos os parágrafos e substituimo-los por \n;
- Na presença de \ substituimo-lo por \\.

Também definimos alfumas funções para nos ajudar com o código:

```
peunome :: [String] -> String
peunome [] = []
peunome (x:xs) = (unwords [head (words x),last (words x)])
```

Esta funçaõ dá como resultado o primeiro e último nome de cada participante necessários para o preenchimento do crachá.

Seguidamente, criamos funções que dão como resultado os campos necessários para para o preenchimento dos crachás. De acordo com cada tipo de inscrição, estas funções dão como resultado o primeiro e último nome, curso e universidade para aluno; primeiro e último nome e universidade para universitário e primeiro e útlimo nome e empresa para empresa. De notar que nos alunos o resultado são 3 Strings, por isso foi conveniente juntar curso e universidade, para isso criamos uma função que os junta numa só string e os separa por um hífen. (Ex: LEI - Universidade do Minho)

E, juntamos as várias funções para cada tipo de inscrição numa só:

```
latexlista :: String -> [[String]]
latexlista k = latexalunos11 k ++ latexalunosext k ++ latexempresas k
```

Para finalizar esta tarefa, criamos três funções que geram o código em LaTeXque, posteriormente, vão originar o pdf os crachás devidamente preenchidos.

1. Primeiro definimos a função `gerarcodigo1` que gera os crachás para todos os participantes, já com o primeiro e último nome, curso-Universidade/Universidade ou Empresa.

   ```
   gerarcodigo1 :: [[String]] -> [[String]]
   gerarcodigo1 [] = []
   gerarcodigo1 (x:xs) = ["\\begin{minipage}{89mm} \n \\includegraphics{design/logo}
   \\\\ \n \n \\addvspace{5mm} \n \n \\begin{center} \n \\huge{" ++ head x ++ "} \n
   \\scriptsize{ \n \\begin{tabular*}{0.75\\textwidth}{c} \n \\hline \n \\end{tabular*}}
   \\\\ \n" ++ last x ++ "\n \\end{center} \n \n \\begin{flushright} \n
   \\begin{tabular}{r l l} \n %\\normalsize{\\color{light-gray}{Participante}} & &
   \n \\normalsize{Participante} & & \n \\end{tabular} \n \\end{flushright}
   \n \\end{minipage} \n \n "] : (gerarcodigo1 xs)
   ```

2. Posteriormente, criamos a função `gerarcodigo2` que define o espaço entre os diversos crachás e coloca 5 crachás por coluna.

3. Por fim, foi definida a função `gerarcodigogeral` que gera o código geral do LaTeX, onde junta as duas funções acima e dá como resultado o ficheiro pdf com os crachás.

   ```
   gerarcodigogeral :: String -> String
   gerarcodigogeral [] = []
   gerarcodigogeral k = "\\documentclass[twocolumn]{article} (...) \\begin{document} \n
   \\AddToShipoutPicture{\\BackgroundPic{design/10pag}} \n "
   ++ gerarcodigo2 (gerarcodigo1 (latexlista k)) ++ " \n \\end{document} \n"
   ```

Mesmo para terminar esta tarefa, criamos duas funções: uma que cria um ficheiro com o nome "Latex.tex" usando o código da função `gerarcodigogeral`, usando a função `writeFile`:

```
gerarcodigo :: String -> IO ()
gerarcodigo k = do writeFile "Latex.tex" (gerarcodigoger
```

E outra, que compila o ficheiro "Latex.tex", criando o respectivo ficheiro pdf:

```
pdf2 :: t -> IO ExitCode
pdf2 k = do system "pdflatex Latex.tex"
```

4 Tarefa 3 e 4

As tarefas 3 e 4 foram trabalhadas em conjunto sendo o meio de publicação das estatísticas pedidas, em ambas as tarefas, um documento LaTeX, com os respectivas tabelas e gráficos trabalhados no gnuplot. Vamos apresentar os vários passos necessários até ao resultado final:

4.1 Definir funções

Primeiramente começamos por definir funções, funções que seguem os seguintes pontos:

- Inscritos por curso

```
mi :: [[String]] -> [[String]]
mi [] = []
mi (x:xs) = if (last (take 2 x)) == "MI" then x: mi xs else mi xs
```

  Como exemplo, apresentamos a função `mi` cujo resultado são apenas os alunos de MI. Esta função, juntamente com as outras correspondentes aos outros cursos (LEI,LCC, MEI, MERSCOM, MBIO), vão ser úteis, no futuro, quando forem criadas as tabelas e os respectivos gráficos.

- Inscritos para o almoço/jantar

```
almocon :: [[String]] -> [[String]]
almocon [] = []
almocon (x:xs) = if (last (take 9 x)) == "0" then x : almocon xs else almocon xs
```

  Aqui definimos funções, como a `almocon/jantarn`, que apresentam todos participantes não inscritos para o almoço e jantar. Por outro lado, definimos funções, como `almocos/jantars`, que apresentam todos os partipantes inscritos para o almoço e jantar.

- Razoẽs das inscrições

```
emp :: [[String]] -> [[String]]
emp [] = []
emp (x:xs) = if (last (take 10 x)) == "Procura 1\186 emprego" || (last (take 11 x))
        == "Procura 1\186 emprego" || (last (take 12 x)) == "Procura 1\186 emprego"
        then x: emp xs else emp xs
```

  Nesta parte criamos funções relativas às razões de inscrição de cada participante. Estas funções indicam os particapantes que se inscreveram nas join por procura do 1º emprego, como é o exemplo da função `emp`, descrita em cima. Mas, também, foram definidas funções para os participantes que escolheram as outras possíveis razões, semelhantes à função `emp`.

4.2 Gnuplot

Posteriormente, trabalhamos com o gnuplot para a apresentação dos nossos dados de forma gráfica para a tarefa 3 e 4.

Começamos por criar ficheiros com os valores que queremos pôr em cada gráfico que iremos criar, de notar por exemplo que, no gráfico apresentado referente ao numero de inscrições, usamos a função pré-definida lenght para fazer a contagem e a função também pré-definida show para trasformar um Int numa String.

```
gnuplotn1 :: String -> String
gnuplotn1 k = show (length (lei (latexalunos k))) ++ " \n" ++ show (length (lcc
(latexalunos k))) ++ " \n" ++ show (length (mi (latexalunos k))) ++ " \n" ++
show (length (mei (latexalunos k))) ++ " \n" ++ show (length (merscom
(latexalunos k))) ++ " \n" ++ show (length (mbio (latexalunos k))) ++ " \n" ++
show (length (latexalunosext k)) ++ " \n" ++ show (length (latexempresas k))

gnuplotn :: String -> IO ()
gnuplotn k = do writeFile "barras1.txt" (gnuplotn1 k)
```

Esta segunda função `gnuplotn` cria o ficheiro "barras1.txt" formado pelo resultado da função `gnuplotn1`. Sendo assim, foram criados três ficheiros: um com o número de inscritos por curso, outro com o número de inscritos para o almoço/jantar e, por fim, outro com o número de inscritos pelas várias razões de inscrição.
Seguidamente criamos três ficheiros: c_barras.txt, c_barras1.txt e c_barras2.txt, em que cada um vai ler o ficheiro correspondente, criado anteriormente. Estes ficheiros apresentam a seguinte estrutura:

```
set terminal push
set terminal latex
set output "graficon.tex"
set boxwidth 1 absolute
set style fill solid 1.0 border -1
set style data histogram
set style histogram cluster gap 1
set yrange [0:20]
set xrange [-0.5:7.5]
set xtics('LEI' -0.04,'LCC' 1, 'MI' 2, 'MEI' 3, 'Merscom' 4,
 'MBIO' 5, 'Externos' 6, 'Empresas' 7)
set ylabel ''
set xlabel ''
plot 'barras1.txt' using 1 t ''
```

Este ficheiro(c_barras1.txt) lê o ficheiro "barras1.txt" através do comando "plot 'barras.txt' using t " " e dá como output o ficheiro "graficon.tex", que será usado mais para a frente.

Para finalizar o trabalho, relacionado com o gnuplot, criamos mais três funções que vão compilar os ficheiros c_barras.txt, c_barras1.txt e c_barras2.txt, originando o ficheiro em LaTeXjá com o código para os respectivos gráficos.

```
gnuplot2 :: t -> IO ExitCode
gnuplot2 k = do system "gnuplot 'c_barras1.txt'"
```

Apresentamos esta função como exemplo, que vai compilar o ficheiro "c_barras1.txt", originando o ficheiro em LaTeXcom o código do gráfico correspondente, definido como "graficon.tex".

4.3 Tabelas

Como já foi referido, o resultado final da tarefa 3 e 4 será um documento LaTeXconstituído pelas tabelas e pelos graficos com os dados trabalhados até aqui. Nesta subsecção vamos explicar como criamos as tabelas e, posteriormente, o documento LaTeX.

Inicialmente, criamos um documento LaTeXcom o código que gera as três tabelas: tabela do número total de inscritos, tabela do número total de inscritos para o almoço/jantar e a tabela das razões de inscrição dos inscritos.

Depois de criado o código, definimos uma função para cada tabela, ou seja, criamos uma função que lê o código correspondente à tabela do número total de inscritos para o almoço/jantar, outra que lê o código correspondente à tabela do número total de inscritos e assim sucessivamente. Esta última é apresentada em baixo, onde o número de alunos, alunos externos e empresas inscritos nas join, vão ser gerados por funções já definidas anteriormente.

```
tablen :: String -> String
tablen k = "\\begin{table}[hl] (...) \\textbf{N\250mero Inscritos} &"
++ show (length (lei (latexalunos k))) ++ "&" (..) \\end{table}"
```

Depois de criadas as três funções referidas e, já com o código, que gera o documento LaTeXfinal, criado, definimos a função `estatistica1` que lê o respectivo código. Nesta função, foi adicionado as funções que geram as diferentes tabelas.

Para concluir o documento final, só falta inserir os gráficos resultantes do gnuplot. Mas, estes já foram inseridos no código através do comando "\input" e dentro de um corpo flutuante.

```
estatistica1 :: String -> String
estatistica1 k = "(...) \\begin{figure}[h]\n \\centering\n \\input{graficoja}
\n \\caption{Tabela do n\250mero de inscritos almo\231o/jantar}\n \\end{figure}(...)"
```

Finalizando, criamos as funções que criam e compilam o ficheiro "Estatistica1.tex".

```
estatistica :: String -> IO ()
estatistica k = do writeFile "Estatistica1.tex" (estatistica1 k)

pdf1 :: t -> IO ExitCode
pdf1 k = do system "pdflatex Estatistica1.tex"
```

5 Conclusão

Perante uma lista de inscrições o nosso programa irá desenvolver por etapas o que nos foi inicialmente prposto:

Tarefa 1 Ao aplicar a função `inscvalidas` à lista de inscições obtemos um ficheiro `Inscricoes.txt` com as inscrições válidas devidamente limpas e editadas e ainda as inscrições mal preenchidas.

Tarefa 2 Ao aplicar a função `gerarcodigo` e `pdf2` à lista de inscições obtemos um ficheiro `Crachas.tex` com e um respectivo PDF com os crachás devidamente preenchidos de cada inscrição válida. De notar que nesta tarefa serão sempre produzidos um numero multiplo de 10 de crachás, ou melhor, perante 8 inscrições válidas serão produzidos 10 cráchas mas apenas 8 serão preenchidos. Esta foi uma das maiores dificuldades que encontramos na realização do trabalho.

Tarefa 3 e 4 Ao aplicar a função `gnuplotn/gnuplotja/gnuplotr` e `gnuplot1/gnuplot2/gnuplot3` temos como resultado os 3 gráficos referentes às 2 tarefas em formato .tex. Por fim aplicando a função `estatistica` e `pdf1` à lista de inscições obtemos um ficheiro `Estatistica.tex` e respectivo PDF com os gráficos e tabelas das 2 tarefas.