1.º Trabalho Prático

Métodos de Programação I
LESI/LMCC
Universidade do Minho

Ano Lectivo de 2001/2002

1 Preâmbulo

Este trabalho deve ser realizado por grupos com um máximo de três alunos. O trabalho deve ser entregue até ao dia 31 de Outubro na Recepção do *Departamento de Informática* (ext. 4430) e nele deve constar a listagem do código desenvolvido assim como um pequeno relatório.

2 Introdução

Nos sistemas da família UNIX (*eg.* LINUX) existe a noção de "shell" de interpretação de comandos ao nível do sistema operativo. Há várias "shells" desse tipo, sendo a *bash* uma das mais populares.

Neste trabalho pretende-se emular em HASKELL a abertura de uma "shell" de programação desse tipo. Uma tal "shell", uma vez invocada, deverá abrir um interpretador de comandos capaz de dar a ilusão de se estar a trabalhar em UNIX/LINUX. É claro que essa ilusão será tanto maior quanto mais extensa e detalhada for a emulação.

Tal como é habitual nos trabalhos desta disciplina, fornece-se de seguida um "kit" para arranque do projecto. O enunciado concluir-se-á com sugestões para valorização do trabalho a realizar.

3 Material fornecido como "kit" para o projecto

Entre na página da disciplina e descarregue, descomprimindo-o (*eg.* via `unzip`), o ficheiro *`mpi0102mp.zip`* que contém o respectivo material pedagógico. Para além de outros ficheiros relevantes para a disciplina, deverá obter:

1. `mpi0102t1.lhs` - trata-se do ficheiro que está a ler neste momento, escrito em "*literate* HASKELL". Isto significa que:

   - se o carregar no HUGS, este interpretador carregará o código HASKELL nele contido e interpretá-lo-á. Sugestão: invoque o HUGS e experimente `:l mpi0102t1.lhs`.

- se o processar via LaTeX (ou PDFLaTeX) obterá este mesmo documento em *PDF*. Sugestão: experimente `pdflatex mpi0102t1.lhs`.
  (Se não está habituado a LaTeX: em LINUX, faz parte da distribuição standard, é só experimentar; em Windows, que tal instalar o *MiKTeX*? Mas note que esse esforço, embora aconselhável, não é essencial à realização deste trabalho.)

2. `mpi0102t1.sty` - trata-se de um ficheiro importado por `mpi0102t1.lhs`, contendo funções de processamento de texto expressas em sintaxe LaTeX (como poderá ver, é normal programar *funcionalmente* em LaTeX).
3. `mpi0102t1.pdf` - trata-se do resultado do processamento de `mpi0102t1.lhs` para sua conveniência, caso não tenha LaTeX acessível.

4 Utilização do "kit" do projecto

Após carregar `mpi0102t1.lhs` no seu interpretador de HASKELL, digite `shell` e entrará no emulador:

```
Main> :l mpi0102t1.lhs
Reading file "mpi0102t1.lhs":

Hugs session for:
/home/fln/share/hugs/lib/Prelude.hs
mpi0102t1.lhs
Main> shell
$
```

Note o novo "prompt" `$`. Para sair, escreva `exit`:

```
$ exit
 :: IO ()
Main>
```

Observe assim o recurso à mónada `IO`. Volte a entrar e tente o bem conhecido comando `cat`:

```
Main> shell
$ cat
Comando desconhecido
$
```

Pois é, esse comando ainda não está previsto em `mpi0102t1.lhs`. E se tentar, por exemplo, `wc`, obterá

```
Main> shell
$ wc
Comando wc nao implementado
$
```

Aqui verifica-se que o comando está já previsto, mas ainda por implementar. Em boa verdade, `mpi0102t1.lhs` é uma *concha* de interpretação de comandos praticamente vazia. Apenas o comando `head` está (parcialmente!) implementado:

```
$ head -1 mpi0102t1.lhs
\documentclass{article}
$
```

O seu trabalho deverá consistir, precisamente, em encher `shell` de conteúdo apropriado. Vejamos então o que já está feito.

5 Análise do "kit" do projecto

Segue-se o código HASKELL que se propõe como ponto de partida para este trabalho.

5.1 Tipos

Observe que `Comando` apenas prevê alguns dos comandos mais conhecidos da *bash*:

```
data Comando = Wc Opcoes [FilePath]
             | Head Opcoes [FilePath]
             | Tail Opcoes [FilePath]
             | Cut  Opcoes [FilePath]
             | Desconhecido String

type Opcoes = String
```

Poderá (e deverá) enriquecer esta *sintaxe abstracta* com mais construtores, por forma a enriquecer o "vocabulário" da sua *shell*.

5.2 Leitura de comandos

Analise com atenção as duas funções que se seguem:

```
readComando :: String -> Comando
readComando l = let [(c,cs)] = lex l
                in case c of
                   "wc"   -> mkCom Wc cs
                   "head" -> mkCom Head cs
                   "tail" -> mkCom Tail cs
                   "cut"  -> mkCom Cut cs
                   _      -> Desconhecido l

mkCom :: (Opcoes -> [FilePath] -> Comando) -> String -> Comando
mkCom c s = case (words s) of
            []          -> c "" []
            (('-':op):r) -> c op r
            (h:t)       -> c "" (h:t)
```

5.3 Execução de Comandos

Esta é realizada pela função monádica `executa`:

```
executa :: Comando -> IO ()
executa (Wc   op fichs) = putStrLn "Comando wc nao implementado"
executa (Head op fichs) = sequence_ (map (rHead (read op)) fichs)
executa (Tail op fichs) = putStrLn "Comando tail nao implementado"
executa (Cut  op fichs) = putStrLn "Comando cut nao implementado"
executa (Desconhecido _)= putStrLn "Comando desconhecido"
```

Note que apenas o comando `head` está (parcialmente) implementado:

```
rHead :: Int -> FilePath -> IO ()
rHead n f = do  s <- readFile f ;
                sequence_ (map putStrLn (take n (lines s)))
```

De facto, o comando não faz qualquer análise de erros e o *output*, quando invocado com múltiplos *file paths*, não corresponde exactamente ao que um sistema UNIX produz.

5.4 Shell interactiva

Finalmente, a função que implementa a abertura da sua *concha* de programação:

```
shell = do  putStr prompt;
            c <- getLine ;
            if c == "exit"
               then stop
               else do  executa (readComando c) ;
                        shell



stop :: IO ()
stop = return ()

prompt = "$ "
```

6 Sugestões para Valorização

- Redefina o tipo `Comando` de forma a poder expressar *pipe*s.
- Ao invés de escrever `Comando desconhecido`, tente invocar um comando equivalente do próprio sistema operativo subjacente.
- Emule em `mpi0102t1.lhs` um número expressivo dos comandos da *bash*.