

Projecto Integrado de Laboratórios de Informática I (UC8201N6)

Universidade do Minho
Licenciatura em Engenharia Informática (LEI)
Ano lectivo de 2012/13

Novembro de 2012

Resumo

Este projecto tem como objectivo consolidar uma visão integrada do uso da linguagem **Haskell** para programação funcional, da *shell* do sistema operativo **Unix** para interação com sistemas de ficheiros e do sistema **L^AT_EX** para preparação de documentação de aplicações de software. Do projecto consta o desenvolvimento de um sistema de software (uma aplicação Web) de pequena escala.

Conteúdo

1	Objectivos	1
2	Aplicações Web	2
3	Material	2
4	O que há a fazer	3
4.1	Instalação e teste	3
4.2	Análise do material disponível	3
4.3	Programação	4
4.4	Relatório e defesa	6
A	Guia de instalação	6
B	Demo	8
C	Esqueletos das funções a implementar	11

1 Objectivos

Com este projecto pretende-se uma consolidação dos conhecimentos aprendidos nas aulas desta disciplina e de outras deste semestre, nomeadamente nas Programação Funcional, por forma a dar aos alunos uma visão integrada do uso dos recursos de programação que aí foram estudados: a *shell* do **Unix**, o sistema **L^AT_EX** para preparação de texto e a linguagem **Haskell**.

O projecto parte de uma base (“kit”) já desenvolvida pela equipa docente da disciplina ¹, convidando os alunos a estudar código escrito por outrem e a usar ferramentas desenvolvidas por terceiros. O objectivo é simular o desenvolvimento de um projecto real de software, onde é frequentemente necessário lidar com a manutenção e integração de código já desenvolvido.

Do tema escolhido — a construção e administração de serviços prestados por aplicações Web — resulta também do trabalho uma acção pedagógica na sensibilização dos alunos para esta importante área da produção de software, encarada na globalidade das unidades curriculares que ensinam programação nesta licenciatura como uma actividade disciplinada, sistemática e fortemente construtiva.

2 Aplicações Web

O projecto consiste essencialmente na extensão e manutenção de uma aplicação Web integralmente construída na linguagem de programação **Haskell**. A aplicação — designada **O_X^{UM}** — é um micro sistema de leilões, em que os utilizadores podem afixar itens para venda, fazer ofertas, etc.

Tal como se verá na descrição que é dada na próxima secção do material que se põe à disposição dos alunos, a maior parte do serviço prestado por esta aplicação já se encontra implementado. O objectivo do trabalho é corrigi-lo, melhorá-lo e acrescentar-lhe novas funcionalidades.

3 Material

O presente documento (enunciado) deve ser cuidadosamente estudado pelos alunos antes de estes iniciarem qualquer esforço de programação neste trabalho. O material disponível para a sua realização obtém-se descompactando o ficheiro

li11213mp.zip

do qual emergem os seguintes ficheiros,

- `Projecto.pdf` — o enunciado que está a ler
- `WebApp.hs` — biblioteca escrita em **Haskell**
- `Auctions.lhs` — biblioteca escrita em **Haskell**
- `Db.hs` — biblioteca escrita em **Haskell**

que vão ser necessárias para fazer o projecto, bem como:

- `Projecto.lhs` — a fonte **L^AT_EX** do ficheiro anterior, pre-processável por **lhs2tex**
- `Referencias.bib` — ficheiro **bibT_EX** necessário para produzir `Projecto.pdf`
- `directoria imagens` onde estão guardadas as imagens deste texto.

¹Os sete membros da equipa docente aproveitam para agradecer a colaboração preciosa de Nuno Carvalho na preparação do “kit” deste projecto.

4 O que há a fazer

4.1 Instalação e teste

A primeira parte do projecto consiste em:

1. Instalar a partir da *shell* as ferramentas a usar, de acordo com as instruções do apêndice A.
2. Seguir as instruções do apêndice B que prefazem uma demo do serviço que já está montado.

A instalação acima referida decorrerá numa *installation party* que será feita na primeira aula de apoio ao projecto.

A demo que se refere é muito importante para os alunos perceberem o significado das operações que já estão disponíveis.

4.2 Análise do material disponível

A figura 1 descreve a arquitectura do sistema que o material da secção 3 disponibiliza, composta por três camadas, a saber:

- camada **W**(eb) — o portal;
- camada **M**(iddleware) — software intermédio, onde são realizadas as principais operações do serviço;
- camada **D**(ata) — o nível dos dados (ligação ao sistema de ficheiros).

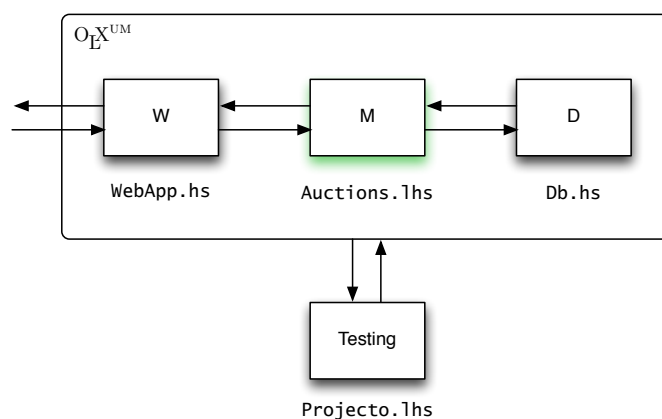


Figura 1: Arquitectura do sistema web .

A segunda parte do trabalho consiste em fazer uma análise do material que é fornecido, a saber:

1. Estudar a biblioteca `Auctions.lhs` — começar por correr, na *shell*

```
lhs2tex Auctions.lhs > Auctions.tex ; pdflatex Auctions
```

produzindo-se assim o ficheiro `Auctions.pdf` que deverá ser impresso e cuidadosamente estudado.

2. Noutra *shell*, executar

```
ghci Auctions.lhs
```

para assim se perceber que código e documentação coincidem, isto é, estão no mesmo ficheiro ².

4.3 Programação

Esta é a parte central do trabalho. As questões que se seguem referem-se ao módulo *Auctions*, a que os alunos deverão acrescentar as funções pretendidas. Os testes unitários que acompanham cada alínea, que devem ser corridos para testar a correção das funções, referem as seguintes configurações de leilões,

```
badhouse =  
  House [b1, b2, b4] [b3] where  
    b1 = Auction 1 "Peter" "TV" 60 ""  
    b2 = Auction 4 "Mary" "laptop" 120 ""  
    b3 = Auction 3 "John" "phone" 85 "Peter"  
    b4 = Auction 3 "Anne" "car" 8500 "Peter"  
  
badhouse1 =  
  House [b1, b2, b4] [b3] where  
    b1 = Auction 1 "Peter" "TV" 60 ""  
    b2 = Auction 4 "Mary" "laptop" 120 ""  
    b3 = Auction 3 "John" "phone" 85 "Peter"  
    b4 = Auction 1 "Peter" "TV" 60 ""
```

para além de *myhouse* que vem em `Auctions.lhs`. Se se correr

```
ghci Projecto
```

(ie., o ficheiro que se está a ler) ver-se-á que os testes estão já implementados neste ficheiro, usando funções “fantasma”, ie. incompletas. Estas, que permitem já a compilação do módulo ³, devem ir sendo apagadas à medida que as soluções pretendidas se forem escrevendo.

1. Implementar a função $idsOk :: House \rightarrow Bool$ que verifica que não há identificadores repetidos (nas duas listas). Testes unitários:

```
test01 = idsOk myhouse  $\equiv$  True  
test02 = idsOk badhouse  $\equiv$  False  
test03 = idsOk badhouse1  $\equiv$  False
```

Correr estes testes para a função que se definiu.

2. Acrescentar a função $idKey :: House \rightarrow Bool$ que verifica se o identificador *actid* de um dado item o identifica univocamente. Testes unitários:

```
test04 = idKey myhouse  $\equiv$  True  
test05 = idKey badhouse  $\equiv$  False  
test06 = idKey badhouse1  $\equiv$  True
```

Correr estes testes para a função que se definiu.

² Esta estratégia designa-se programação literária [Knu92] e o seu princípio base é o seguinte: *a documentação de um programa deve coincidir com ele próprio*. Por outras palavras, o código fonte e a sua documentação deverão constar do mesmo documento (ficheiro). O texto que se está a ler é um exemplo de programação literária.

³Ver último anexo.

3. Será que *idKey* implica *idsOk*? Ou será que *idsOk* é que implica *idKey*? Ou nenhuma dessas implicações se verifica? Justifique a sua resposta.
4. Acrescentar o predicado *allIds* :: *House* → *Bool* que verifica se a alocação de identificadores a leilões é sequencial. Testes unitários:

test07 = *allIds myhouse* ≡ *True*
test08 = *allIds badhouse* ≡ *False*

Correr estes testes para a função que se definiu.

5. É fácil de ver, exercitando OX^{UM} , que a função *auctionBid* não está implementada convenientemente, pois:
 - após fazer **Reset**, se licitar o item 1 por 61 euros e depois por 1 (um) euro, verá que o sistema aceita qualquer oferta, mesmo que seja inferior à melhor oferta até ao momento, violando o princípio do *quem dá mais ganha* inerente a qualquer leilão;
 - se licitar um item que não existe, por exemplo o item 99, o sistema dá erro.

Reescreva a função por forma a nenhuma das anomalias identificadas acima se verificar.

6. Supondo que o formato interno *House* muda para

data *NHouse* = *NHouse* { *tot* :: [*NAuction*] } **deriving** (*Show*, *Eq*)

onde

data *NAuction* = *NAuction* { *a* :: *Auction*, *st* :: *Status* } **deriving** (*Show*, *Eq*)
data *Status* = *Running* | *Finished* **deriving** (*Eq*, *Show*)

escrever funções de conversão entre os dois formatos,

toNHouse :: *House* → *NHouse*
toHouse :: *NHouse* → *House*

Testes unitários:

test09 = *toHouse (toNHouse myhouse)* ≡ *myhouse*
test10 = *toNHouse (toHouse h)* ≡ *h* **where** *h* = *toNHouse badhouse*

Correr estes testes para a função que se definiu.

7. Redefinir o tipo *Auction* acrescentando-lhe mais um campo — *actclass*,

data *Auction* = *Auction* {
actid :: *Int*,
actowner :: *String*,
actdesc :: *String*,
actclass :: *String*,
actvalue :: *Int*,
actbidder :: *String*
} **deriving** (*Show*, *Eq*, *Ord*)

que classifica itens em categorias (eg. electro-doméstico, mobiliário) e escrever uma nova função de administração que totaliza os valores leiloados por categoria (*x* de electro-domésticos, *y* de mobiliário, etc).

8. Um dos defeitos do modelo de dados *Auction* é não guardar o valor inicial de licitação. Corrija esta limitação, criando ainda uma função que contabilize quanto é que, em média, os itens vendidos se valorizaram no leilão.
9. Ver que funções foram definidas que podem ser generalizadas polimorficamente, redefinindo-as com o seu tipo mais geral.

Valorização. Escolher entre as seguintes sugestões:

1. Como operação de manutenção, pretende-se repor um item não vendido no leilão, especificando nova base de licitação. Implementar esta operação.
2. O módulo *Auctions* oferece a função genérica *sortOn* para ordenar sequências de registos segundo um dos seus atributos. Use-a de forma a que a opção *List* de O_X^{UM} mostre leilões ordenados por identificador.

4.4 Relatório e defesa

Deste trabalho deverá ser feito um pequeno relatório que deverá ser entregue ao monitor do turno de cada grupo, bem como o código desenvolvido, tudo integrado num único ficheiro processável por *lhs2tex*.

Oral. Cada grupo deverá defender o seu relatório numa apresentação oral em data e moldes a definir na *página* da disciplina.

Anexos

A Guia de instalação

Este anexo contém as instruções de instalação das ferramentas necessárias para correr o “kit” do projecto, a saber:

- Ubuntu

1. Instalar GHCi (<http://www.haskell.org/platform/linux.html>)

```
$ sudo apt-get install haskell-platform
```

2. Actualizar o Cabal

```
$ cabal update
$ cabal install cabal-install
```

3. Adicionar o novo cabal à PATH com precedência em relação à versão disponível para todos os utilizadores do sistema

```
$ echo 'export PATH=~/.cabal/bin:$PATH' >> ~/.bashrc
```

4. Usar Cabal para instalar o Happstack:

```
$ cabal install primitive
$ cabal install aeson
$ cabal install happstack-lite
```

5. Instalar lhs2TeX:

```
$ sudo apt-get install texlive-full
$ cabal install lhs2tex
```

Como teste, regenerar o ficheiro que se está a ler:

```
$ lhs2TeX Project.lhs > Project.tex  
$ pdflatex Project.tex
```

- Mac OS X

1. Instalar GHCI (<http://www.haskell.org/platform/mac.html>)
2. Corrigir a path do Cabal para o gcc ⁴.
3. Actualizar o Cabal

```
$ cabal update  
$ cabal install cabal-install
```

4. Actualizar o *link* simbólico para o binário do Cabal (adaptar de acordo)

```
$ sudo rm /usr/bin/cabal  
$ sudo ln -s ~/Library/Haskell/ghc-7.0.3/lib/ \  
cabal-install-1.16.0.1/bin/cabal /usr/bin/cabal
```

5. Usar Cabal para instalar o Happstack:

```
$ cabal install aeson  
$ cabal install happstack-lite  
$ cabal install happstack-server
```

6. Instalar lhs2TeX:

```
$ cabal install lhs2tex
```

Como teste, regenerar o ficheiro que se está a ler:

```
$ lhs2TeX Project.lhs > Project.tex  
$ pdflatex Project.tex
```

- Windows

1. Instalar GHCI (<http://www.haskell.org/platform/windows.html>)
2. Instalar Cabal (<http://www.haskell.org/cabal/download.html>)

```
> cd C:\Program Files\Haskell Platform\2012.4.0.0\bin
```

3. Actualizar o Cabal

```
> cabal.exe update  
> cabal.exe install cabal-install  
> cd C:\Documents and Settings\Afonso\Application Data\cabal\bin
```

4. Usar Cabal para instalar o Happstack

```
> cabal.exe install aeson  
> cabal.exe install happstack-lite-7.2.0
```

5. Definir a porta do Happstack Server

```
main = simpleHTTP nullConf { port = 80 } $ handlers
```

6. Instalar MiKTeX (<http://miktex.org/>) ⁵

7. Adicionar a directoria onde se encontra o pdflatex à PATH

⁴<http://tinyurl.com/c6w2ks9>

⁵Nota: no Windows XP pode dar erro a finalizar a instalação mas os executáveis encontram-se no sistema.

```
Control Panel > System > Advanced > Environment Variables  
System Path C:\Program Files\MiKTeX 2.9\miktex\bin
```

8. Definir o environment code para poder compilar o PDF do enunciado (http://www.haskell.org/haskellwiki/Literate_programming)

```
\newenvironment{code}{\verbatim}{\endverbatim}  
\newenvironment{spec}{\verbatim}{\endverbatim}
```

9. Como teste, regenerar o ficheiro que se está a ler:

```
> pdflatex Project.lhs
```

B Demo

Uma vez feita a instalação tal como se descreve no apêndice A, é recomendável executar as acções seguintes na directoria em que o ficheiro `li11213mp.zip` foi descompactado, a saber:

1. Ao nível da *shell*, executar

```
ghci WebApp
```

2. De seguida, dentro do interpretador, correr a função `main`.
3. O interpretador, que fica aparentemente bloqueado, deverá ter iniciado o serviço da aplicação web `OLXUM`: para se interagir com este, há que arrançar um qualquer *browser* e nele abrir o endereço

```
http://localhost:8000
```

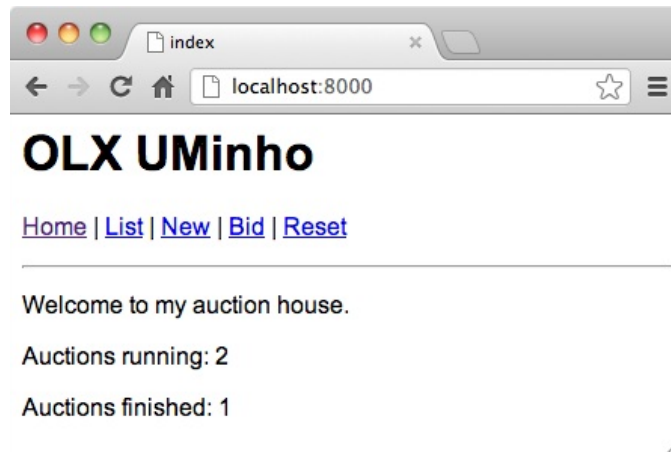


Figura 2: Leilão acabado de abrir.

Dever-se-á obter uma página com o aspecto da figura 2.

De seguida, há que reproduzir cuidadosamente a sequência de operações que se segue, na interface da aplicação web:

1. Carregar em `List` e verificar que o aspecto da janela do *browser* passa a ser o da figura 3, em que se podem ver os artigos que estão em leilão e os que já foram leiloados ou não tiveram comprador.

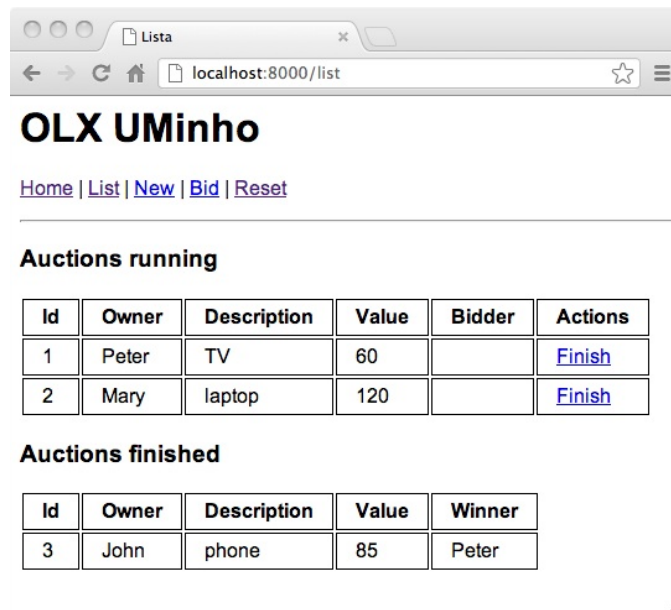


Figura 3: Estado corrente do leilão.

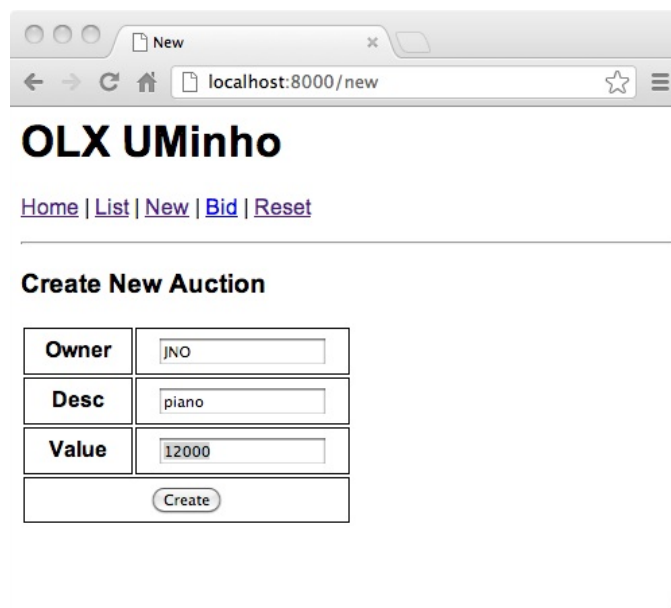
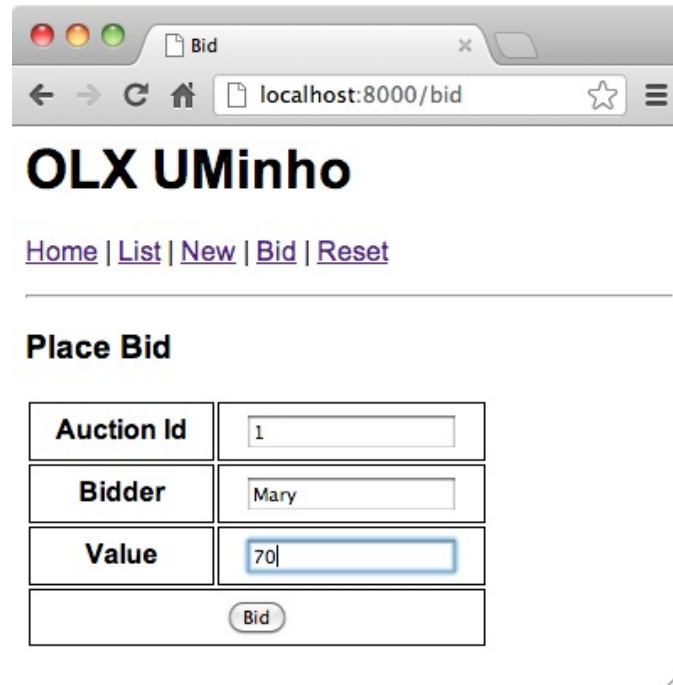


Figura 4: Entrada de um novo artigo no leilão.

2. Seleccionar a opção New e proceder como se mostra na figura 4 para registar um novo artigo para ser leilado.
3. Seleccionar a opção Bid e, tal como se mostra na figura 5, registar uma oferta de Mary para o artigo nr.1 que Peter tem no leilão.
4. Simular a aceitação por parte de Peter da oferta de Mary carregando em Fi-

nished na correspondente linha. O resultado de List passa a ser o dado na figura 6.



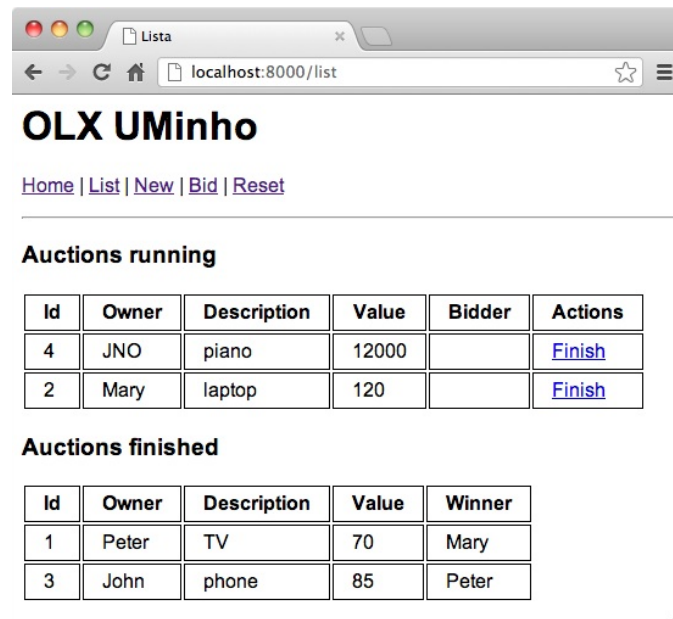
OLX UMinho

[Home](#) | [List](#) | [New](#) | [Bid](#) | [Reset](#)

Place Bid

Auction Id	<input type="text" value="1"/>
Bidder	<input type="text" value="Mary"/>
Value	<input type="text" value="70"/>
<input type="button" value="Bid"/>	

Figura 5: Uma nova oferta para um artigo em leilão.



OLX UMinho

[Home](#) | [List](#) | [New](#) | [Bid](#) | [Reset](#)

Auctions running

Id	Owner	Description	Value	Bidder	Actions
4	JNO	piano	12000		Finish
2	Mary	laptop	120		Finish

Auctions finished

Id	Owner	Description	Value	Winner
1	Peter	TV	70	Mary
3	John	phone	85	Peter

Figura 6: Uma oferta aceite no leilão (Mary compra a TV que Peter tinha posto a leilão).

5. Finalmente, carregar em **Reset** e de novo em **List**, para se verem os dados iniciais repostos.

Desta forma se fica a perceber como funciona o leilão O_X^{UM} do ponto de vista do *utilizador* desta aplicação no web.

Administração. Contudo, há operações no leilão que não estão visíveis aos utilizadores. Tratam-se de operações internas, administrativas, a que não se quer dar acesso ao utilizador, por exemplo: fazer a contabilidade de quanto se vendeu até ao momento, a dos itens que não foram vendidos, etc.

Operações deste género são feitas pelos administradores do sítio “por trás da cortina”. É o que o leitor pode agora fazer, abrindo outra *shell* Unix noutra janela, sobre a mesma directoria, e nela invocando, de novo:

```
ghci WebApp
```

De seguida:

1. Como exemplo de execução de operações em *modo de administração*, sugere-se que se corra, nesse interpretador em “background”:

```
*Main> evalOnWebData tots
Tot {tfin = 85, trun = 180, tlost = 0}
*Main>
```

Esta operação calcula os totais seguintes: *tfin* dá o total já leiloado; *trun* dá o total a ser leiloado; *tlost* dá o total que não foi leiloado.

2. Para se ver como os modos de utilização e de manutenção operam concorrentemente, volte-se ao website e repita-se a oferta de Mary da figura 5. Se na *shell* de administração se repetir

```
*Main> evalOnWebData tots
Tot {tfin = 85, trun = 190, tlost = 0}
*Main>
```

ve-se-á que o total correntemente a ser leiloado aumentou $70 - 60 = 10$ euros.

3. Suponha-se agora que, de novo no *website*, Mary não tem ofertas para o seu *laptop* e desiste do leilão, carregando em **Finished**. Agora a mesma operação do modo de administração,

```
*Main> evalOnWebData tots
Tot {tfin = 85, trun = 70, tlost = 120}
*Main>
```

passará a contabilizar 120 euros como valor perdido (não lícitado), correspondente ao *laptop* que se não vendeu.

Sugere-se que se façam mais experiências do género das descritas acima antes de se abordar a parte central do trabalho.

C Esqueletos das funções a implementar

```
idsOk :: House → Bool
idsOk h = error "idsOk: Por implementar"
```

```

idKey :: House → Bool
idKey h = error "idKey: Por implementar"

allIds :: House → Bool
allIds h = error "allIds: Por implementar"

toHouse :: NHouse → House
toHouse h = error "toHouse: Por implementar"

toNHouse :: House → NHouse
toNHouse h = error "toNHouse: Por implementar"

pKey :: (Auction → Int) → [Auction] → Bool
pKey a l = error "pKey: Por implementar"

```

Referências

- [Knu92] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.