# Linguagens e Ambientes de Programação (2018/2019)

## Prática 4

Módulos. Input/output. Interação com o utilizador. Exercícios 28 e 29.

- 28 - Desenvolva em OCaml um módulo chamado `BinTree` onde sejam representadas e manipuladas **árvores binárias de inteiros**.

O nome do tipo exportado deve ser

```
tree
```

A representação interna deve ser a seguinte:

```
type tree = Nil | Node of int * tree * tree ;;
```

O módulo deve ser fechado ou seja deve ocultar a representação interna das árvores e das funções auxiliares (se houver). Repare que internamente se poderia usar representações diferentes: por exemplo árvores n-árias em que se guardava em cada nó interno apenas duas sub-árvores.

Eis a lista de funções públicas pretendidas. Tente fazer as primeiras quatro durante a aula.

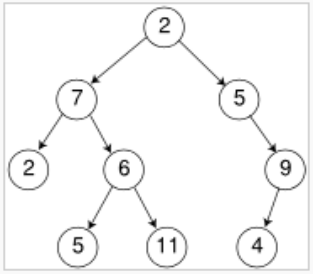
```
make : int list -> tree          (* Cria uma árvore com os elementos da lista, alinhados para a direita  *)
max : tree -> int                (* Determina o valor maior que ocorre numa árvore *função parcial) *)
load : string -> tree            (* Carrega uma árvore a partir dum ficheiro de texto *)
store : string -> tree -> unit   (* Escreve uma árvore num ficheiro de texto *)
show : tree -> unit              (* Mostra uma árvore na consola *)
```

Exemplos:

```
make [1;2;3] = Node (1, Nil, Node (2, Nil, Node (3, Nil, Nil)))
max (make [1;2;3]) = 3
```

**Representação das árvores em ficheiro e no ecrã**

Considere a seguinte árvore:




A função `store` escreve no ficheiro, primeiro a raiz, depois a sub-árvore esquerda, depois a sub árvore direita. A árvore vazia é escrita como "-".

```
2
7
2
-
-
6
5
-
-
11
-
-
5
-
9
4
```

```
-
-
-
```

A função `show` apresenta na consola a árvore da forma abaixo, usando indentação. A informação que resulta da indentação permite omitir a escrita dos dois "-", no caso das folhas.

```
2
  7
    2
    6
      5
      11
  5
    -
    9
      4
      -
```

**Como criar o módulo "BinTree" no Eclipse**

- `Project > Automatic Build` - Confirme que esta opção está ativa.
- `File > New > OCaml Project (ocamlbuild)` - crie um projeto novo chamado "Aula4".
- `File > New > Module "BinTree"` - cria o ficheiro BinTree.ml.
- `Project > Properties > Project` ... escrever na primeira caixa "BinTree.byte", e fazer OK.
- O módulo, mesmo que vazio, já ficou compilado e internamente foram gerados os ficheiros "BinTree.cmi" e "BinTree.cmo" (não se observam, mas estão lá).
- Escreva o código do módulo BinTree.ml. Para já, trata-se dum modulo aberto.
- Agora, para ocultar a representação da árvore num módulo fechado, use o botão da direita do rato sobre a janela BinTree.ml e escolha o comando `Generate interface`. O ficheiro "BinTree.mli" é criado. Neste ficheiro, altere a definição do tipo tree para este ficar opaco e faça "Save" para o módulo ser recompilado.
- Para carregar o módulo BinTree fechado no interpretador, use o botão da direita do rato sobre a janela BinTree.ml e escolha o comando `Load in Top Level`.
- Experimente as seguintes expressões no Ocaml Toplevel, por exemplo usando da forma habitual a tecla F6:

```
open BinTree ;;
make [1;2;3;4;5] ;;
- : BinTree.tree = <abstr>
```

**Como criar o módulo "BinTree" usando a consola**

- Use um editor de texto para criar os ficheiros BinTree.ml e BinTree.mli.
- Compile o módulo BinTree assim

```
ocamlc -c BinTree.mli BinTree.ml
```

  e são gerados os ficheiros BinTree.cmi e BinTree.cmo.
- Experimente na consola:

```
$ ocaml
        Objective Caml version 3.12.1

# #load "BinTree.cmo" ;;
# open BinTree ;;
# make [1;2;3;4;5] ;;
- : BinTree.tree = <abstr>
```

---

- 29 - Desenvolva um programa interativo e **standalone** que afixe no ecrã dados obtidos a partir de ficheiros contendo árvores de inteiros. Use o módulo desenvolvido no exercício anterior. Chamamos **ficheiro com árvore** a um ficheiro de texto que contém uma árvore no formato descrito no exercício anterior.

Os comandos que o programa deve suportar são os seguintes:

- mostra *fich* - mostra a árvore do ficheiro *fich* na consola (usando `load` e `show`);
- maximo *fich* - mostra qual o maior valor que ocorre na árvore do ficheiro *fich*;
- ajuda - apresenta a lista de comandos disponíveis;
- sair - abandona o programa.

**O que é um programa standalone?**

É um programa OCaml concebido para ser compilado e para funcionar com independência do interpretador `ocaml`. Depois de compilado, um programa standalone pode ser invocado diretamente a partir do prompt do Unix, tal como outro programa qualquer. Quem usar o programa não sabe se ele foi escrito em OCaml, Java, C, ou C++, etc.

Um programa standalone define-se como uma sequência de funções terminada pela chamada duma função que se designa por *função principal* (muitas vezes dá-se o nome de `main` à função principal). Um programa standalone tem a responsabilidade de gerir todo o seu input e output, pois não tem o interpretador `ocaml` a ajudar.

**Como gerar o programa standalone "Main" no Eclipse**

Relativamente ao que já se fez antes, fazer adicionalmente:

- `File > New > Module "Main"` - cria o ficheiro Main.ml.
- Escreva o conteúdo do ficheiro Main.ml.
- `Project > Properties > Project` ... apague o que está na primeira caixa, escrever "Main.byte", e fazer OK.
- Para correr o programa, use o botão da direita do rato sobre Main.byte e escolher:

```
Run As > Ocaml Executable
```

**Como gerar o programa standalone "Main" na consola**

- Usar um editor de texto para criar os ficheiros BinTree.mli, BinTree.mlie Main.ml.
- Para compilar e gerar o executável, use o comando (a ordem dos argumentos é importante):

```
ocamlc -o Main BinTree.mli BinTree.ml Main.ml
```

- Para correr o programa:

```
./Main
```

**Ajuda**

A maior parte deste exercício já está feito. Estude-o, para aprender a organizar um programa interativo em ML, e complete-o. Falta só terminar a função `exec`.

Note que a função `split` é um exemplo de aplicação do método indutivo a strings. A chamada recursiva é sobre o argumento `xs`, que representa a "cauda" da string.

```
open BinTree ;;  (* Abre o modulo das árvores *)

(* Método indutivo aplicado a strings. A função cut "separa a cabeça da cauda", numa string. *)
let cut s =  (* pre: s <> "" *)
    (String.get s 0, String.sub s 1 ((String.length s)-1))
;;
(* Método indutivo aplicado a strings. A função join adiciona um char à cabeça numa string. *)
let join x xs =
    (Char.escaped x)^xs
;;
let rec split s =                      (* parte a string s no primeiro ' ' e produz um par ordenado de strings *)
    if s = "" then ("", "")            (* primeiro caso base *)
    else
      let (x,xs) = cut s in            (* separa cabeça da cauda *)
         if x = ' ' then ("", xs)      (* segundo caso base *)
         else let (a,b) = split xs in  (* caso geral - chamada recursiva para a cauda *)
             (join x a, b)
;;
let help () =
    print_string "Comandos válidos:\n" ;
    print_string "    mostra fich\n" ;
    print_string "    maximo fich\n" ;
    print_string "    ajuda\n" ;
    print_string "    sair\n"
;;
let byeBye () =
    print_string "Até à vista!\n";
    exit 0
;;
let exec comm filename = (* falta apenas completar esta funcao *)
    match comm with
        "mostra" -> ()
      | "maximo" -> ()
      | "ajuda" -> help ()
      | "sair" -> byeBye ()
      | _ -> help ()
;;
let error mesg =
    output_string stderr mesg ;
    output_string stderr "!\n" ;
    flush stderr
;;
let rec main () = (* ciclo de interpretacao *)
```

```
        (try
            print_string "> " ;
            let line = read_line () in
                let (comm, fileName) = split line in
                   exec comm fileName
        with
           End_of_file -> byeBye ()
         | Sys_error str -> error str
         | _ -> error "Erro") ;
        main ()
    ;;

    main () ;; (* Esta linha faz o programa começar a correr aqui *)
```

---

- 30 -

a) Escreva uma função indutiva

```
countEmpty: string -> int
```

que, dado o nome dum ficheiro, conte o número de linhas vazias (linhas com zero caracteres) nesse ficheiro.

b) Escreva uma função indutiva

```
clear: string -> string -> int
```

que, dados os nomes dum ficheiro de entrada e dum ficheiro de saída, copie o primeiro ficheiro para o segundo, mas omitindo as linhas vazias (linhas com zero caracteres). A função deve ainda retornar o número de linhas vazias que forem omitidas durante a cópia.

---