Mestrado Integrado em Engenharia Informática (FCT/UNL)
Ano Letivo de 2014/2015
Linguagens e Ambientes Programação – Teste 1

11 de Abril de 2015 às 09:30

Open book test with duration of 1 four e 30 minutes + 15 minutes leeway

Nome:	Num:

Notes: *This exam paper consists of three groups of questions. Answer in the exam paper itself, using the front and the back..*
In the OCaml problems show that you know how to use the inductive method and write, if possible, type 1 or 2 functions..
Do not use imperative mechanisms and reasoning and also do not simulate imperative mechanisms and reasoning.
Pode definir funções auxiliares sempre que quiser e também pode usar funções do módulo `List` *do OCaml.*
You can define auxiliary functions whenever you want and can also use functions of the List module.
Normally, imperfect answers deserve some points.
Fraud implies failure in the course.

1. [3 valores] Multiple choice. Wrong answers do not discount. Provide the answers here:

A	B	C

A) For the OCaml and C languages, which is the only true statement?

a) Although the C language is older than the OCaml language, the first official standard of C is more recent than the first official standard of OCaml.

b) Considering the factors "execution speed" and "memory consumption", C is more effective than OCaml even when running recursive functions.

c) The functional part of the OCaml and the C involve almost irreconcilable different views to the problem-solving process.

d) In OCaml the computation is usually based on the use of side effects, whereas in C the computation is not usually based on the use of side effects

B) Interpreters and compilers. What is the true statement?

a) Considering the sequence of actions a compiler performs, type checking normally comes before parsing to avoid spending time parsing of expressions that, after all, have invalid types.

b) Running an already compiled program requires that the compiler is installed on the same machine.

c) Compilers do not run programs. To run a program, we need some kind of interpreter.

d) In general, to run a program written in a high level language, we first need to process this program using a compiler.

C) For the following function with three arguments, which of the following statements is true?

```
let rec doIt f l z =
    match l with
        [] -> z
      | x::xs -> f x (doIt f xs z)
;;
```

a) The function **f** is binary and right-associative, being used to combine to a single value all the value in the list **l**.

b) The function **f** is binary and left-associative, being used to combine to a single value all the value in the list **l**.

c) If the argument list **l** is empty, then the result is **z**. If the argument list **l** is not empty, then the result is not **z**.

d) Ultimately, the function **doIt** applies the function **f** only to the first and last element of **l**, to obtain the result. The remaining elements of the list l end up not contributing to the result.

2. [3 valores] What is the OCaml type of the following function:

```
let rec f a b = if a then [a] else a::b ;;
```

3. We can use n-ary trees to represent structured text documents with logical divisions such as chapters, sections, sub-sections, etc. In this group of problems, we will use the **doc** type, to represent structured texts:

```
type α ntree = NNil | NNode of α * α ntree list ;;
type doc = string ntree ;;
```

As an example, here is a portion of the book "Alice's Adventures in Wonderland." We only show the first three lines of each of the first two chapters, but it already a good enough example.

```
let alice =
   NNode("Alice's Adventures in Wonderland, by Lewis Carroll", [
          NNode("CHAPTER I - Down the Rabbit-Hole", [
                NNode("Alice was beginning to get very tired...",[]);
                NNode("peeped into the book her sister was reading, but...",[]);
                NNode("Alice 'without pictures or conversations?'",[])
          ]);
          NNode("CHAPTER II - The Pool of Tears", [
                NNode("'Curiouser and curiouser!' cried Alice...",[]);
                NNode("English); 'now I'm opening out like the largest telescope....",[]);
                NNode("seemed to be almost out of sight, they were getting so far...",[])
          ]
   ]);;
```

In the **doc** representation doc, each **logical division** is described by a node that records in the *value* the **division title** and records in the *children list* the lower level divisions. In every logical division, the children list should never be empty

The text lines are stored in the leaves of the tree. The lines are not logical divisions, but physical as they are exclusively related to the text layout on paper. Each **line** is represented by a node that records in the *value* the text, with an always empty child list. What allows us to detect that a node represents a line is the fact that its children list is empty.

The only types of logical division that occur in the example of Alice are "complete document" and "chapter", but it is normal a document to have more types of logical division

a) [3.5 points] A valid document is a document where the length of each title and of each line never exceeds 80 characters. [To determine the length of a string use the function `String.length`]. Write a function to check whether a document is valid.

```
validate: doc -> bool
```

Examples

```
validate NNil = true
validate (NNode("Hello!", []) = true
validate alice = true
```

b) [3.5 points] It is customary to identify the logical divisions of a document using a sequence of numbers separated by periods. For example, `2.1` marks the first section of the second chapter of a document. In the Alice example there are no sections, so `2.1` identify the first line of the second chapter. To represent an integer sequence, such as `2.1`, we use a list of integers: [2: 1] in this case.

Write a function that, given a document, gets the particular division of the document that is identified by a list of integers. If the list does not identify any division, then the result is `NNil`.

```
val get : doc -> int list -> doc
```

Examples

```
get NNil [1;3] = NNil
 get alice [1;3] =
    NNode("Alice 'without pictures or conversations?'", [])
```

c) [3.5 points] We want to print a document in a continuous roll of paper. Of course, every document text line will occupy a line on paper. But, be aware, the titles of logical divisions are printed prominently and using more generous vertical spacing: a title occupies exactly the equivalent to three normal lines.

Write a function to determine the amount of paper required to print a document. The amount of paper is measured in lines.

```
val print_length : doc -> int
```

Examples

```
print_length NNil = 0
print_length alice = 15
```

d) [3.5 points] If we choose to use single sheets instead of continuous paper, our printing system can only handle documents containing special ASCII character 12, called *form feed*, which forces the printer move to the next sheet. We want to write a function to insert this special character at the beginning of some titles and the beginning of some lines of the document. We know that each page has a capacity of 60 lines and we want to ensure that nothing is printed beyond that limit. We also want to make good use of the vertical space each sheet, writing as many lines as possible in it. As in the previous problem, assume that each title occupies the vertical space corresponding to 3 normal lines.

The *form feed* character is written in OCaml as follows, within a string: `"\012"`.

Write a function to paginate a document, by producing a modified copy of it.

```
paginate: doc -> doc
```

Hint: You can solve the problem as you like. Nevertheless, a good technique is to use an auxiliary function with the following functionality: the function receives the document and the line number where the print head is initially positioned (a value from 0 to 60); the function returns the document already paged and the line number on which the print head is positioned at the end (a value from 0 to 60).

```
pag: doc -> int -> doc * int
```

Example, imagining that the capacity of each page is only 5 lines. Pay attention to all the `\012`.

```
paginate alice =
     NNode("Alice's Adventures in Wonderland, by Lewis Carroll", [
          NNode("\012CHAPTER I - Down the Rabbit-Hole", [
              NNode("Alice was beginning to get very tired...",[]);
              NNode("peeped into the book her sister was reading, but...",[]);
              NNode("\012Alice 'without pictures or conversations?'",[])
          ]);
          NNode("CHAPTER II - The Pool of Tears", [
              NNode("'Curiouser and curiouser!' cried Alice...",[]);
              NNode("\012English); 'now I'm opening out like the largest telescope....",[]);
              NNode("seemed to be almost out of sight, they were getting so far...",[])
          ]
     ]);;
```