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

Teste com consulta com 1 hora e 30 minutos de duração + 15 minutos de tolerância

Nome:	Num:

Notas: *Este enunciado é constituído por 3 grupos de perguntas. Responda no próprio enunciado,* ***usando a frente e o verso****.*
Nos problemas em OCaml mostre que sabe usar o método indutivo e escreva, se possível, funções de tipo 1 ou 2.
Não use mecanismos ou raciocínios imperativos nem simule mecanismos ou raciocínios imperativos.
Pode definir funções auxiliares sempre que quiser e também pode usar funções do módulo `List` *do OCaml.*
Normalmente, respostas imperfeitas merecem alguma pontuação.
Fraude implica reprovação na cadeira.

1. [3 valores] Escolha múltipla. As respostas erradas não descontam. Indique as respostas aqui:

A	B	C

A) Relativamente às linguagens OCaml e C, qual é a única afirmação verdadeira?

a) Apesar da linguagem C ser mais antiga do que a linguagem OCaml, a primeira norma oficial do C é mais recente do que a primeira norma oficial do OCaml.

b) Considerando os fatores "velocidade de execução" e "consumo de memória", o C é mais eficaz do que o OCaml, mesmo a executar funções recursivas.

c) A parte funcional do OCaml e a linguagem C envolvem visões diferentes, quase irreconciliáveis, do processo de resolução de problemas.

d) Em OCaml a computação baseia-se normalmente no uso de efeitos laterais, enquanto que em C a computação não se baseia normalmente no uso de efeitos laterais.

B) Interpretadores e compiladores. Qual é a afirmação verdadeira?

a) Na sequência de ações que um compilador executa, a verificação de tipos vem normalmente antes da análise sintática para se evitar perda tempo a fazer a análise sintática das expressões que afinal têm tipos inválidos.

b) Para conseguir correr um programa já compilado é preciso que o compilador esteja instalado na mesma máquina.

c) Os compiladores não executam programas. Para correr um programa, precisamos de algum tipo de interpretador.

d) De forma geral, para executar um programa escrito numa linguagem de alto nível, precisamos primeiro de processar esse programa usando um compilador.

C) Relativamente à seguinte função com três argumentos, qual das seguintes afirmações é a verdadeira?

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

a) A função **f** é binária e associativa à direita, sendo usada para combinar todos os valores da lista **l** num valor único.

b) A função **f** é binária e associativa à esquerda, sendo usada para combinar todos os valores da lista **l** num valor único.

c) Se a lista argumento **l** for vazia, o resultado é **z**. Se a lista argumento **l** não for vazia, o resultado é diferente de **z**.

d) Em última análise, a função **doIt** aplica a função **f** apenas ao primeiro e ao último elemento de **l** para obter o resultado final. Os restantes elementos da lista **l** acabam por não contribuir para o resultado.

2. [3 valores] Diga qual o tipo OCaml da seguinte função:

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

3. Podemos usar árvores n-árias para representar documentos de texto estruturados, com divisões lógicas, tais como capítulos, secções, subsecções, etc. Neste grupo de problemas, usaremos o tipo **doc**, para representar textos estruturados:

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

Para exemplificar, eis uma porção do livro "Alice's Adventures in Wonderland". Só se mostram as três primeiras linhas de cada um dos dois primeiros capítulos, mas já serve de exemplo.

```
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...",[])
          ]
   ]);;
```

Na representação **doc**, cada **divisão lógica** é descrita por um nó que regista no *valor* o **título da divisão** e cuja *lista de filhos* guarda divisões de nível inferior. Em qualquer divisão lógica, a lista de filhos nunca deve ser vazia.

As linhas de texto ficam guardadas nas folhas da árvore. As linhas não são divisões lógicas, mas sim físicas, pois estão exclusivamente relacionadas com a disposição do texto no papel. Cada linha é representada por um nó que regista no *valor* o **texto**, sendo a *lista de filhos* sempre vazia. O que nos permite detetar que um nó representa uma linha é o facto da sua lista de filhos ser vazia.

Os únicos tipos de divisão lógica que ocorrem no exemplo da Alice são "documento completo" e "capítulo", mas é normal um documento ter maior riqueza de tipos de divisão lógica.

a) [3.5 valores] Um documento válido é um documento onde o comprimento de cada título e de cada linha não excede os 80 carateres. [Para determinar o comprimento duma string use a função `String.length`]. Escreva uma função para verificar se um documento é válido.

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

Exemplos

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

b) [3.5 valores] É habitual identificar as divisões lógicas do documento usando uma sequência de números separados por pontos. Por exemplo, `2.1` assinala a primeira secção do segundo capítulo dum documento. No exemplo da Alice não existem secções, pelo que `2.1` acaba por identificar a primeira linha do segundo capítulo. Para representar uma sequência de inteiros, como a `2.1`, usamos uma lista de inteiros: `[2;1]` neste caso.

Escreva uma função que, dado um documento, obtenha a divisão identificada por uma lista de inteiros. Se a lista de inteiros não identificar nenhuma divisão, então o resultado é `NNil`.

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

Exemplos

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

c) [3.5 valores] Queremos imprimir um documento num rolo de papel contínuo. Naturalmente, cada linha de texto do documento vai ocupar uma linha no papel. Mas, atenção, os títulos das divisões lógicas são impressos com destaque e usam espaçamento vertical mais generoso: um título ocupa exatamente o equivalente a três linhas normais.

Escreva uma função para determinar a quantidade de papel necessária para imprimir um documento, usando como unidade de medida a linha normal.

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

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

d) [3.5 valores] Se optarmos por usar folhas soltas em vez de papel contínuo, o nosso sistema de impressão só consegue processar documentos que incluam o caráter especial ASCII 12, chamado *form feed*, que força a impressora passar à folha seguinte. Queremos escrever uma função para inserir esse caráter especial no início de alguns títulos e no início de algumas linhas do documento. Sabemos que cada página tem capacidade para 60 linhas e queremos de garantir que nunca se imprime para lá desse limite. Também queremos aproveitar bem o espaço vertical cada folha, escrevendo nela o maior número possível de linhas. Tal como no problema anterior, assumimos que cada título ocupa o espaço vertical correspondente a três linhas normais.

O caráter *form feed* escreve-se assim em OCaml, dentro duma string: `"\012"`.

Programe então uma função para paginar um documento, que produza uma cópia modificada.

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

Ajuda: Pode resolver o problema como desejar. Mas uma boa técnica consiste em usar uma função auxiliar com a seguinte funcionalidade: a função recebe o documento e o número da linha em que a cabeça da impressora está posicionada inicialmente (um valor de 0 a 60); a função retorna o documento já paginado e o número de linha em que a cabeça da impressora fica posicionada no final (um valor de 0 a 60).

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

Exemplo, imaginando que a capacidade de cada página é de apenas 5 linhas. Olhe bem para os `\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...",[])
          ]
    ]);;
```