Licenciatura em Engenharia Informática (FCT/UNL)
Ano Lectivo 2007/2008
Linguagens e Ambientes Programação – Época de Recurso

23 de Julho de 2008 às 17:00

Exame com consulta com 2 horas e 45 minutos de duração + 15 minutos de tolerância

Nome:	Num:

Notas: *Este enunciado é constituído por 6 perguntas e 9 folhas. Responda no próprio enunciado.*
Nos problemas em ML mostre que sabe usar o método indutivo.
Pode definir funções/métodos auxiliares sempre que precisar (muitas vezes é mesmo preciso)
Fraude implica reprovação na cadeira.

1. [2 valores] Para cada frase, escreva na zona sublinhada se ela é VERDADEIRA ou FALSA (ou use as abreviaturas V e F):

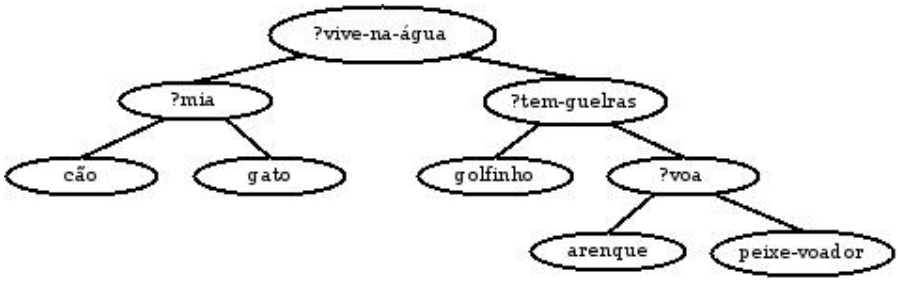
- É possível verificar em tempo de compilação se as expressões dum programa qualquer em ML, C ou C++ estão conformes ao sistema de tipos de cada linguagem (ou seja, livres de erros de tipo). _____
- Num qualquer programa em ML, livre de erros de tipo, garante-se que a aplicação duma operação primitiva aos seus argumentos tem sempre sucesso, durante a execução do programa. _____
- No interior duma função C++ que tenha dois parâmetros inteiros, um passado por valor e outro passado por referência, esses dois parâmetros nunca podem representar a mesma zona de memória. _____
- No interior duma função C++ que tenha um parâmetro inteiro passado por referência, esse parâmetro e uma variável global nunca podem representar a mesma zona de memória. _____
- Numa linguagem com escopo estático é possível determinar em tempo de compilação se cada variável declarada é efectivamente usada em alguma expressão. _____
- Numa linguagem com escopo dinâmico é possível determinar em tempo de compilação se cada variável declarada é efectivamente usada em alguma expressão. _____
- Em JavaScript é possível acrescentar em tempo de execução um novo membro (membro de dados ou membro funcional) a um objecto individual, ou a todos os objectos do mesmo tipo. _____
- Em C++ é possível acrescentar em tempo de execução um novo membro (membro de dados ou membro funcional) a um objecto individual, ou a todos os objectos do mesmo tipo. _____
- Em C, uma função com um argumento de tipo double também poder ser aplicada a inteiros. Isto é um exemplo de polimorfismo de inclusão. _____
- Num projecto de programação em C/C++ podem escrever-se os ficheiros ".c" e ".cpp" sem ter de tomar qualquer medida especial, pois o ligador consegue lidar com os vários ficheiros objecto ".o" gerados pelo compilador. _____

2. [2 valores] Diga qual o tipo da seguinte função em ML:

```
let rec f l =
    match l with
        [] -> 0
      | (a,b)::xs -> a (b (f xs))
;;
```

3. **Árvores de decisão binárias em ML**. O conceito de árvore de decisão binária já é bem conhecido do projecto 1 de LAP. Uma árvore de decisão binária é constituída por **questões** (nós internos binários) e por **entidades** (as folhas da árvore). A árvore de decisão vazia não está prevista. Nesta pergunta vamos assumir que todas as funções se aplicam a árvores consistentes, ou seja, a árvores sem dados contraditórios.

Eis um exemplo duma árvore de decisão binária que representa algum conhecimento sobre animais:



O tipo das árvores de decisão binárias pode definir-se assim em ML:

```
type dtree = Entity of string | Question of string * dtree * dtree ;;
```

Resolva os seguintes três problemas sobre árvores de decisão. Tratam-se de problemas não relacionados, ou seja, nenhum deles ajuda a resolver os restantes.

a) [2 valores] Escreva em OCaml uma função

```
rightmost: int -> dtree -> string
```

que, dado um valor inteiro positivo e dada uma árvore de decisão binária, descubra o nó Entity mais à direita, cujo conteúdo (uma string) tenha o comprimento indicado. A função produz esse conteúdo se o encontrar. Caso contrário devolve a string vazia "". Exemplos:

```
rightmost 4 (Entity "gato") = "gato"
rightmost 3 (Entity "gato") = ""
rightmost 4 (Question "?mia" (Entity "gato") (Entity "rato")) = "rato"
rightmost 3 (Question "?mia" (Entity "cão") (Entity "gato")) = "cão"
rightmost 2 (Question "?mia" (Entity "cão") (Entity "gato")) = ""
```

[Ajuda: Para determinar o comprimento duma string, use a função função `String.length`.]

b) [2.5 valores] Escreva em OCaml uma função

```
cmp: dtree -> dtree -> itree
```

para comparar duas árvores de decisão binárias. O resultado da comparação é uma árvore binária de inteiros do seguinte tipo:

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

A forma mais simples de explicar qual o resultado da função é a seguinte: Imagine que as duas árvores-argumento são desenhadas em dois acetatos transparentes e que esses acetatos são sobrepostos. A árvore-resultado tem a mesma estrutura da sobreposição, sendo o valor de cada um dos nós da árvore-resultado determinado pelas seguintes regras:

- Para dois nós Entity sobrepostos, o valor do nó correspondente na árvore-resultado é 1 se as entidades forem iguais e 0 se as entidades forem diferentes.
- Para dois nós Question sobrepostos, o valor do nó correspondente na árvore-resultado é 1 se as questões forem iguais e 0 se as questões forem diferentes.
- Para um nó Question e um nó Entity sobrepostos, o valor do nó correspondente na árvore-resultado é -1.
- Para um nó sem correspondente na outra árvore, portanto sem sobreposição, o valor do nó correspondente na árvore-resultado é -2.

Exemplos:

```
cmp (Entity "gato") (Entity "gato") = Node(1,Nil,Nil)
cmp (Entity "cão") (Entity "gato") = Node(0,Nil,Nil)
cmp (Question("?mia", Entity "cão", Entity "gato"))
     (Question("?mia", Entity "cão", Question("?mia", Entity "cão", Entity "gato")))
            = Node(1, Node(1, Nil, Nil), Node(-1, Node(-2,Nil,Nil), Node(-2,Nil,Nil)))
```

[Ajuda: É possível aplicar emparelhamento de padrões a duas árvores ao mesmo tempo, assim:

```
match t1, t2 with
  Entity e1, Entity e2 -> …
| Entity e, Question(q,n,y) -> …
… etc. …
… etc. …                          ]
```

c) [2.5 valores] Escreva em OCaml uma função

```
dm: string -> dtree -> int
```

que, dada uma questão e uma árvore de decisão binária, determine a distância mínima na árvore entre duas ocorrências dessa questão. Assume-se que árvore-argumento nunca tem mais do que 1000 nós e convenciona-se que a resposta 1001 significa que a questão não ocorre na árvore pelo menos duas vezes (no fundo, 1001 representa a *distância infinita*).

O que é exactamente a distância mínima entre dois nós? É o número mínimo de arcos da árvore que é preciso atravessar para ir de um nó ao outro. Muitas vezes é preciso subir primeiro na árvore até um nó ascendente comum, e depois descer para alcançar o segundo nó. Exemplos:

```
dm "?mia" (Question("?mia", Entity "cão", Entity "gato")) = 1001
dm "?mia" (Question("?mia", Entity "cão", Question("?mia", Entity "cão", Entity "gato"))) = 1
```

 Nome: Num:

4. [2 valores] Considere o seguinte programa escrito em GCC, uma variante do C que suporta aninhamento de funções:

```
#include <stdio.h>

#define SIZE   4
int v[SIZE] = {90, 91, 92, 93} ;

void F(int *a, int n) {
    int v[SIZE] ;
    void Update(void) {
       int i ;
       for( i = 0 ; i < SIZE ; i++ )
          v[i] = a[i] + n ;
    }
    Update() ;
    F(v, n+1) ;
}

int main(void) {
    int x = 1 ;
    F(v, x) ;
    return 0 ;
}
```

Mostre qual o estado da pilha de execução no momento em que nela se encontrem exactamente 5 registos de activação (incluindo o da função `start`).

Para efeitos da criação do registo de activação inicial, imagine que cada programa em GCC está embebido numa função sem argumentos chamada `start`. Depois trate todas as entidades globais do programa como sendo locais à função imaginária `start`. Assuma também que a primeira célula da pilha de execução é identificada como posição 0, a segunda célula da pilha de execução é identificada como posição 1, etc. Estas são as mesmas convenções que foram usadas nas aulas teóricas e práticas.

41	27	13
40	26	12
39	25	11
38	24	10
37	23	09
36	22	08
35	21	07
34	20	06
33	19	05
32	18	04
31	17	03
30	16	02
29	15	01
28	14	00

5. [4 valores] **Sopa de Letras em C++**. Considere o seguinte rectângulo de letras. Repare que nele surgem diversas ocorrências de palavras, por exemplo **`ananas`** e **`aveia`**:

```
asfanananas
tnfdfdsfdfd
anananananaan
ananasabano
aveiaomapae
java sbomnl
ocamlacriat
tragodoxos o
```

O objectivo deste problema é desenvolver um pequeno **sistema de classes** para representar rectângulos de letras e efectuar contagens nesses rectângulos. Nas classes devem ser definidos os construtores, destrutores e mais as duas seguintes operações:

`int Count(char c)` – Conta o número de ocorrências da letra-argumento.
`int Count(char pal[])` – Conta o número de ocorrências da palavra-argumento.

O argumento dos construtores é um array de strings – que se declara assim `char *arg[]` – em que o array é terminado por NULL e cada string é terminada por '\0'. A representação interna nas classes concretas é a mesma do argumento dos construtores. Para poupar tempo, assuma que o valor do argumento já é válido, ou seja tem mesmo a forma dum rectângulo (i.e. todas as linhas têm igual comprimento). Também se considera válido um rectângulo de letras vazio (com tamanho 0x0).

Pedimos-lhe para, implementar completamente as seguintes três classes:

`class CharRect` – Simples "interface" em C++ que introduz as operações pretendidas.
`class CharRectH` – Herda da classe anterior. A função que conta palavras considera apenas palavras sem sobreposição que ocorrem na horizontal da esquerda para a direita.
`class CharRectHV` – Herda da classe anterior. A função que conta palavras considera palavras sem sobreposição que ocorrem na horizontal da esquerda para a direita, mais palavras sem sobreposição que ocorrem na vertical de cima para baixo.

Este é um problema sobre reutilização de código. Mostre que sabe usar bem a linguagem C++, incluindo o seu mecanismo de herança.

6. [3 valores] Use a função genérica de biblioteca do C `qsort`, que ordena arrays, para programar uma função de ordenação de listas de valores reais. O tipo das listas é dado pela seguinte declaração:

```
typedef struct Node {
    double data ;
    struct Node *next;
} Node, *List ;
```

O cabeçalho da função que lhe pedimos para programar em C é o seguinte:

```
List ListSort(List l) ;
```

Cuidado, que os valores guardados na lista podem estar a ser referidos por apontadores externos à lista. Por isso a função `ListSort` tem a responsabilidade de criar a nova lista usando os mesmos nós da lista original e sem lhes mudar o valor - só podem ser mudados os apontadores que se encontram nesses nós e definem a sequência da lista.

[Ajuda: para resolver este problema vai precisar de usar um array auxiliar, cujo tamanho depende do comprimento da lista a ordenar.]

O que se segue é o manual da função `qsort`, para sua orientação. Usámos esta função na nossa aula pática 7.

NAME

qsort - sorts an array

SYNOPSIS

```
#include <stdlib.h>
void qsort(void *base, size_t nmemb, size_t size, int(*compar)(const void *, const void *));
```

DESCRIPTION

The qsort() function sorts an array with **nmemb** elements of size **size**. The **base** argument points to the start of the array.

The contents of the array are sorted in ascending order according to a comparison function pointed to by **compar**, which is called with two arguments that point to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.