Licenciatura em Engenharia Informática (FCT/UNL)
Ano Lectivo 2009/2010
Linguagens e Ambientes Programação – Época Normal

21 de Junho de 2010 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 grupos de perguntas e 7 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] Perguntas de escolha múltipla. Escolha a resposta mais correcta e responda aqui:

A	B	C	D	E

A) Quando um programa OCaml é compilado com sucesso…

a) Não há erros/excepções causadas por anomalias de execução.
b) Quando executado, o programa termina, produzindo um resultado.
c) Quando o programa é executado, não vão ocorrer erros de tipo, apesar de não existir qualquer informação de tipo disponível dinamicamente, inclusivamente em relação às funções polimórficas.
d) A sobrecarga (overloading) passa a poder ser tratada em tempo de execução.

B) "Todas as linguagens com tipificação dinâmica suportam polimorfismo de forma inerente." Esta frase é verdadeira para que espécie de polimorfismo?

a) Polimorfismo paramétrico.
b) Polimorfismo de inclusão.
c) Overloading.
d) Coerção.

C) Relação entre o C e o C++?

a) Todos os programas válidos em C são também programas válidos em C++.
b) Há programas escritos em C++ que também são programas válidos em C.
c) As incompatibilidades entre o C e o C++ residem apenas no tipo **`void *`**.
d) A linguagem C++ respeita o sistema de tipos da linguagem C, limitando-se a adicionar objectos e classes.

D) Numa expressão dum programa Javascript…

a) Nunca há erros sintácticos
b) As conversões de tipo automáticas tornam rara a ocorrência de erros de tipo.
c) Nunca há erros semânticos.
d) A avaliação termina sempre.

E) Regra de escopo dinâmico vs. regra de escopo estático.

a) A regra de escopo dinâmico é usada nas linguagens dinamicamente tipificadas.
b) A regra de escopo estático permite que uma linguagem seja estaticamente tipificada.
c) A regra de escopo dinâmico implica uma implementação mais complexa de funções aninhadas.
d) A regra de escopo estático implica uma implementação mais simples de funções aninhadas.

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

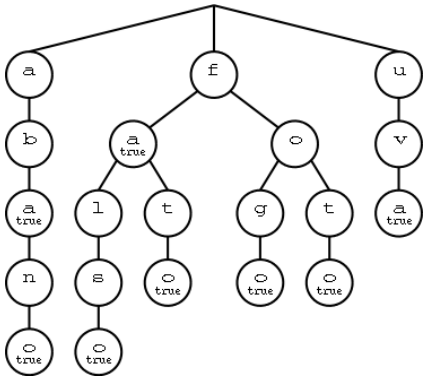
```
let f a b c = a (a b c) 0 ;;
```

3. ***Floresta lexical*** é um tipo de estrutura de dados que se adapta bem à representação de **conjuntos de palavras**. Uma floresta lexical é uma lista de árvores n-árias contendo uma letra em cada nó. Cada palavra é representada como um percurso descendente a partir da raiz de alguma árvore: um valor booleano, existente em cada nó, marca o fim de cada palavra sempre que for igual a `true`. As várias palavras codificadas numa floresta lexical **partilham prefixos**, para se evitar redundância. Além disso, cada lista de árvores está **ordenada crescentemente** pelas letras das raízes e **nenhuma folha** pode conter o valor `false`.

   Concretizando melhor, numa floresta lexical, cada nó tem três elementos: uma letra, um marcador booleano e a lista dos seus filhos. Eis uma representação de florestas lexicais em OCaml, que usa dois tipos mutuamente recursivos:

```
type lexForest = lexTree list
and lexTree = Node of char * bool * lexForest ;;
```

   Um exemplo: A seguinte floresta lexical, constituída por três árvores, representa o seguinte conjunto de palavras: “aba”, “abano”, “fa”, “falso”, “fato”, “fogo”, “foto” e “uva”. Para melhor leitura da floresta, só mostramos os valores booleanos iguais a “true”. Há 8 palavras, logo há 8 ocorrências de `true`.



   Eis o termo ML correspondente a esta floresta:

```
let example =
   [Node ('a', false,
      [Node ('b', false,
         [Node ('a', true, [Node ('n', false, [Node ('o', true, [])])])])]);
   Node ('f', false,
      [Node ('a', true,
         [Node ('l', false, [Node ('s', false, [Node ('o', true, [])])]);
          Node ('t', false, [Node ('o', true, [])])]);
       Node ('o', false,
         [Node ('g', false, [Node ('o', true, [])]);
       Node ('t', false, [Node ('o', true, [])])])]);
   Node ('u', false, [Node ('v', false, [Node ('a', true, [])])])]
```

   Para exemplificar, a função count determina o número total de palavras que ocorrem numa floresta. Note que, no tratamento de cada nó, se torna necessário considerar a lista dos nós filhos “cs” e a lista dos nós irmãos “ts”.

```
let rec count z =
    match z with
        [] -> 0
      | Node(c,b,cs)::ts -> (if b then 1 else 0) + count cs + count ts
;;
```

   Ajuda: os problemas que se sequem podem ser resolvidos usando o método indutivo de forma directa usando funções de tipo 1.

a) [1 valor] Escreva em OCaml uma função, muito simples, para contar o número de ocorrências duma letra numa floresta lexical.

```
countLetter: char -> lexForest -> int
(countLetter 'a' example = 4)
```

b) [2 valores] Escreva em OCaml uma função `union` para efectuar a união de dois conjuntos de palavras.

```
union: lexForest -> lexForest -> lexForest
(union example example = example)
(union example [] = example)
```

Complete o que falta:

```
let rec union z1 z2 =
    match z1, z2 with
          [], _ -> z2
        | _, [] -> z1
        | Node(c1,p1,cs1)::ts1, Node (c2,p2,cs2)::ts2  ->
```

c) [2 valores] Escreva em OCaml uma função `inter` para efectuar a intersecção de dois conjuntos de palavras.

```
inter: lexForest -> lexForest -> lexForest
(inter example example = example)
(inter example [] = [])
```

d) [2 valores] Escreva em OCaml uma função `sort` para ordenar uma floresta lexical que não respeite o critério da ordenação. Esta função converte numa floresta válida uma floresta que falhe o critério da ordenação.

```
sort: lexForest -> lexForest
(sort example = example)
(sort [] = [])
```

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

```
int main(void) {
  int a = 0 ;
  void G(int *pc) { int c = 0 ; *pc += 2 ; main() ; }
  void F(int *pb) { int b = 0 ; *pb += 1 ; G(&b) ; }
  F(&a) ;
  return 0 ;
}
```

Mostre qual o estado da pilha de execução no momento em que acabou de ser empilhado e preenchido o sétimo registo de activação (incluindo os registos das funções `start` e `main`.)

Use as convenções habituais das aulas: 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 00, a segunda célula da pilha de execução é identificada como posição 01, etc.

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. [2 valores] Considere, em ANSI-C, o tipo das **listas ligadas** que foi estudado nas aulas teóricas e usado em alguns exercícios das aulas práticas.

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

Recorde-se que se podem criar nós para estas listas usando a função:

```
List NewNode(Data val, List next)
{
    List n = malloc(sizeof(Node)) ;
    if( n != NULL ) {
       n->data = val ;
       n->next = next ;
    }
    return n ;
}
```

Escreva uma função `merge` que, dadas duas listas **ordenadas sem repetições**, produza uma nova **lista ordenada e sem repetições** com todos os valores distintos que ocorrem nas listas dos argumentos. A lista resultante é constituída por nós novos, independentes dos nós que ocorrem nas listas dos argumentos.

```
List Merge(List l1, List l2) ;
```

É maior valorizada uma solução iterativa (programada usando um ou mais ciclos) do que uma solução baseada em recursividade.

6. [5 valores] Considere o seguinte problema de representação de animais que vivem num zoo para crianças. Eis alguns exemplos de animais que vivem no zoo:

```
Cão - Nome: "Bobby" - Alim. preferido: "carne" - Dentes: 28
Papagaio - Nome: "Louro" - Alim. preferido: "sementes de girassol" - Compr. das asas: 75 cm
Galinha - Nome: "Luísa" - Alim preferido: "milho" - Compr das asas: 90 cm
Galinha - Nome: "Maria" - Alim preferido: "minhocas" - Compr das asas: 80 cm
Papagaio - Nome: "Matias Jr." - Alimento preferido: "alpista" - Compr das asas: 50 cm
Gato - Nome: "Tomilho" - Alim preferido: "peixe cozido" - Dentes: 30
Gato - Nome: "Branquinha" - Alim preferido: "queijo" - Dentes: 29
```

Normalmente, cada animal tem alguns amigos entre os animais do zoo. Todos os dias, cada animal perde um amigo aleatoriamente (caso tenha algum) e também tenta criar uma nova amizade com um animal escolhido aleatoriamente entre todos os animais (se o animal escolhido for o próprio ou se já for amigo, nada acontece). O facto de X ser amigo de Y, não significa obrigatoriamente que Y seja amigo X.

Problema: Escreva um programa que: (1) mostre na consola toda informação respeitante a cada animal que vive no zoo; (2) faça os animais viver um dia e liste na consola todas as mudanças de amizades usando mensagens da forma: "Branquinha estabelece amizade com Louro" e "Bobby quebra amizade com Tomilho".

Importante: No seu programa defina dum sistema de classes abstractas e concretas para representar os animais. Pretende-se uma solução modular e extensível.

Simplificação: Omita do seu programa o tratamento das aves. Além disso, o carregamento dos dados dos animais (a partir de ficheiro) já se assume feito.

Código de partida. Dado o seguinte código de partida, falta programar as classes, num ficheiro ".h" e num ficheiro ".cpp". Não se esqueça de escrever os construtores e destrutores das classes. Todas as classes têm acesso à variável global `all`. Use a função de biblioteca `rand()`, a qual gera valores aleatórios inteiros no intervalo `[0, RAND_MAX]`. Se quiser alterar este código oferecido, tem liberdade de o fazer.

```
#include <iostream>
#include <vector>
#include <stdlib.h>
#include <time.h>

class Animal ;
std::vector<Animal *> all ; // Todos os animais

class Animal {            // A definição de Animal está incompleta…
  public:
     virtual void Show() ;
     virtual void Friendship() ;
} ;

int main() {               // A função main está completa
     LoadAll("animal.txt") ;
     srand(time(NULL)) ;
     for( int i = 0 ; i < all.size() ; i++ )
            all[i]->Show() ;
     for( int i = 0 ; i < all.size() ; i++ )
            all[i]->Friendship() ;
     return 0 ;
}
```

Operações da classe `vector`:

```
int size()
Animal operator[] (int)
void push_back(const Animal *)
void pop_back()
```