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

12 de Julho 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) Foi escrita em C++ uma função `Ln` que calcula o logaritmo natural dum valor real. Apesar da função produzir sempre resultados certos, estranhamente deixa sempre o argumento a zero (caso esse argumento seja uma variável). Qual é a melhor explicação para este problema?

a) O parâmetro está a ser passado por valor e deveria ser passado por referência constante.
b) O parâmetro está a ser passado por referência constante e deveria ser passado por valor.
c) O parâmetro está a ser passado por referência e deveria ser passado por valor.
d) A função foi programada de forma recursiva e deveria, pelo contrário, ser programada usando ciclos.

B) Sobre o sistema de tipos do JavaScript.

a) É uma linguagem com um sistema de tipos seguro.
b) Usa escopo dinâmico.
c) Suporta objectos e classes, tal como o Java.
d) Não é tipificado.

C) Considere em OCaml duas funções aninhadas, definidas usando `let rec`, e com a função G definida dentro da função F:

a) O ambiente interior das duas funções é necessariamente idêntico.
b) O ambiente interior da função G é necessariamente mais vasto (tem mais nomes) do que o da função F.
c) Um mesmo nome pode designar entidades diferentes nos dos ambientes interiores às duas funções.
d) A função G tem necessariamente acesso a todo o ambiente global.

D) Sobre polimorfismo.

a) O C suporta polimorfismo de inclusão.
b) O Javascript não suporta nenhuma forma de polimorfismo.
c) O OCaml suporta polimorfismo paramétrico e polimorfismo de overloading.
d) O C++ e o Java suportam polimorfismo de inclusão, paramétrico, overloading e coerção.

E) Gestão automática de memória está disponível nas seguintes linguagens:

a) C++ e Java.
b) C e OCaml.
c) C e C++.
d) OCaml, Java e Javascript.

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

```
let f (a, b) x = (a x, b x) ;;
```

3. ***Matrizes de números*** - Considere a utilização de **listas de listas** para representar **matrizes rectangulares** em OCaml. Uma matriz é representada pela lista das suas linhas, tendo todas as linhas igual comprimento. A matriz vazia é representada pela lista vazia, sendo proibida qualquer outra representação da matriz vazia. Nos problemas que se seguem, só trabalhamos com matrizes válidas.

   O tipo das matrizes é o seguinte:

```
type α matrix = α list list ;;
```

   Eis um exemplo duma matriz (neste caso de inteiros) com duas linhas e três colunas:

```
let example = [[1; 2; 3];
               [4; 5; 6]] ;;
```

a) [1.5 valores] Escreva em OCaml uma função `count` para contar o número de ocorrências dum dado valor numa matriz.

```
count: α -> α matrix -> int
(count 4 example = 1)
```

   Complete o que falta.

```
let rec count b m =
    match m with
          [] ->

        | []::xs ->

        | (v::vs)::xs ->

;;
```

b) [1.5 valores] Escreva em OCaml uma função `first` para produzir a primeira coluna duma matriz não vazia (a matriz vazia não tem primeira coluna).

```
first: α matrix -> α list
(first example = [1; 4])
(first [[1; 2; 3]] = [1])
```

   Complete o que falta. O caso base da função corresponde a uma matriz que tem lá dentro só uma linha não vazia. No caso geral da função, separa-se a primeira linha das restantes linhas da matriz.

```
let rec first m =
    match m with

          [v::vs] ->

        | (v::vs)::xs ->
;;
```

c) [1 valores] Escreva em OCaml uma função `rest` para produzir o que resta duma matriz, eliminando a primeira coluna. Só faz sentido definir para matrizes não vazias.

```
rest: α matrix -> α matrix
(rest example = [[2; 3];[5;6]])
```

d) [1 valores] Escreva em OCaml uma função `diag` para produzir a diagonal principal duma matriz. A matriz pode ser vazia. Pode usar as funções das alíneas anteriores, mesmo que não as tenha programado.

```
diag: α matrix -> α list
(diag example = [1; 5])
```

e) [1 valores] Escreva em OCaml uma função `trans` para transpor uma matriz, que pode ser vazia. Transpor significa trocar linhas com colunas. Pode usar as funções das alíneas anteriores, mesmo que não as tenha programado.

```
trans: α matrix -> α matrix
(trans example = [[1; 4];
                  [2; 5];
                  [3; 6]])
```

f) [1 valores] Escreva em OCaml uma função `magic` para determinar uma matriz é um **quadrado mágico**: ou seja, se a matriz é quadrada e se a soma de cada linha, de cada coluna e de cada uma das duas diagonais, dá sempre o mesmo resultado. A matriz vazia é mágica. Pode usar as funções das alíneas anteriores, mesmo que não as tenha programado.

```
magic: int matrix -> bool
(magic example = false)
(magic [[1; 1];
        [1; 1] = true)
(magic [[2; 7; 6];
        [9; 5; 1]
        [4; 3; 8]] = true)
```

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

```
#include <stdio.h>

int vect[3] = {11, 22, 33} ;

void F(int *p) {
    void G(int **h) {
       (*h)[1] = (*h)[0] ;
       F(*h + 1) ;
    }
    G(&p) ;
}

int main(void) {
  F(vect) ; // passa o endereço do vector global
  return 0 ;
}
```

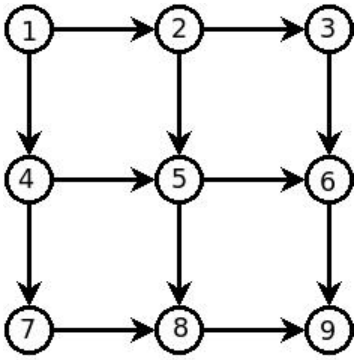
Mostre qual o estado da pilha de execução no momento em que acabou de ser empilhado e preenchido o **quinto** registo de activação (incluindo os registos das funções `start` e `main`.)

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 seguinte tipo que permite definir estruturas dinâmicas em que cada nó pode ser ligado a dois outros nós usando os apontadores `next` e `down`.

```
typedef struct Node2 {
    int data ;
    struct Node2 *next, *down ;
} Node2, *List2 ;
```



Para criar nós, pode usar a seguinte função:

```
List2 NewNode2(int val, List2 next, List2 down) {
    List2 n = malloc(sizeof(Node2)) ;
    if( n != NULL ) {
       n->data = val ;
       n->next = next ;
       n->down = down ;
    }
    return n ;
}
```

Escreva uma função `MakeSquare` que, dado um número inteiro não negativo `n`, construa uma estrutura de nós para representar um quadrado de números com n linhas e n colunas contendo todos os valores inteiros de `1` até n^2, organizados por ordem crescente ao longo das várias linhas. No caso do argumento ser `0`, o resultado deve ser `NULL`. No caso do argumento ser 3, o resultado é a estrutura que se apresenta em cima à direita.

```
List2 MakeSquare(int n)
```

É mais valorizada uma solução iteractiva, programada usando ciclos, do que uma solução baseada em recursividade.

[Sugestão: Para resolver parte do problema, adapte a função `ListMakeRange` da aula teórica 15.]

.

6. [5 valores] **Círculos e esferas em C++.** Um **círculo** é uma figura bidimensional caracterizada por um centro (dado por dois atributos `x` e `y` de tipo `double`) e por um raio (dado por um atributo `radius` de tipo `double`). Uma **esfera** é uma figura tridimensional caracterizada por um centro (dado por três atributos `x`, `y` e `z` de tipo `double`) e por um raio (dado por um atributo `radius` de tipo `double`).

   Neste problema, vamos considerar o tipo das esferas como subtipo dos círculos! Mais concretamente, uma esfera é um círculo com um atributo suplementar `z`, o qual é usado nas operações a 3 dimensões mas é ignorado nas operações a 2 dimensões. Quando se misturam círculos e esferas no mesmo contexto, a 3ª dimensão das esferas é ignorada, sendo tudo tratado a 2 dimensões. A partir de agora, quando se falar em círculos, também estamos a falar de esferas, pois estas são um caso particular.

   Programe completamente, em C++, uma classe concreta `Circle` para representar círculos, mais uma subclasse concreta `Sphere` para representar esferas. As operações pretendidas são as seguintes (pode definir operações auxiliares):

   ```
   Constructores
   Destructores
   bool Inside(const Circle& that)          Testa se that está completamente dentro de this
   static double MinimalCovering(const std::vector<Circle&>& all)
                                    Determina o raio do menor círculo que cobre todos os círculos dados
   ```

   Na função `MinimalCovering` assume-se que os centros de todos os círculos ou esferas são colineares, ou seja, se situam sobre uma mesma recta.

   Importante: As classes `Circle` e `Sphere` devem ser organizadas para que a subclasse beneficie ao máximo do mecanismo da herança.

   Na subclasse `Sphere`, as duas operações que aceitam um círculo como argumento devem usar um cast dinâmico para testar se esse círculo é uma esfera e, no caso de o ser, as operações devem ser efectuadas usando a 3ª dimensão.

   [Ajudas: Para obter o tamanho dum vector use `v.size()`.
   Assuma que estão disponíveis duas funções globais para calcular a distância entre dois pontos, a 2 e a 3 dimensões:

   ```
   double dist2(double x1, double y1, double x2, double y2) ;
   double dist3(double x1, double y1, double z1, double x2, double y2, double z2) ;]
   ```