Licenciatura em Engenharia Informática (FCT/UNL)
Ano Lectivo 2010/2011
Linguagens e Ambientes Programação – Época Normal

16 de Junho de 2011 às 09:00

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

Nome:	Num:

1. [2 valores] Escolha múltipla. As respostas erradas não descontam. Escolha as respostas mais correctas e responda aqui:

A	B	C	D	E

A) Quais dos seguintes conjuntos de linguagens suportam, sob alguma forma, a regra de resolução de nomes chamada de *escopo dinâmico*?

a) C e C++.
b) Java e OCaml.
c) Javascript.
d) Nenhuma das linguagens: C, C++, OCaml, Java, Javascript.

B) Quando se escreve um programa em C numa determinada plataforma computacional e o programa já foi exaustivamente testado e parece funcionar…

a) Se testarmos o programa numa plataforma diferente, é possível que se descubram erros que não foram detetados na plataforma original.
b) Podemos concluir que o programa está correto, mas só temos garantias na plataforma em que foi testado.
c) Se o programa foi escrito de forma independente do tamanho da palavra da máquina (16 bits, 32 bits, etc.), então o programa está correto para funcionar em qualquer plataforma.
d) Se o programa foi escrito de forma independente do tamanho da palavra da máquina (16 bits, 32 bits, etc.) e se não fizer nenhum `malloc`, então o programa está correto para funcionar em qualquer plataforma.

C) Qual a melhor descrição do que faz a seguinte função booleana em C++?

```
bool F(const vector <int> &a) {   // precondition: a is sorted
         for(int k = 1; k < a.size(); k++)
            if(a[k-1] == a[k])
               return true;
         return false;  }
```

a) Retorna sempre true.
b) Confirma se o vetor a está realmente ordenado.
c) Determina se o vetor a contém algum valor repetido.
d) Determina se todos os valores do vetor a são iguais.

D) Ainda relativamente à função anterior, qual a melhor explicação para se ter passado o objecto do argumento por referência constante?

a) O vetor a é alvo da aplicação da operação de indexação, tendo por isso de ser passado por referência.
b) É mais económico passar o objeto por referência do que por valor.
c) Não há qualquer razão especial. Era igual ter sido passado por valor.
d) Trata-se da mera aplicação duma regra puramente convencional em C++: "todos os objetos devem ser passados por referência constante".

E) Sobre a linguagem Javascript, qual destas frases é a mais correta?

a) Foi idealizada para se escreverem programas pequenos e simples, sendo no entanto possível escrever programas grandes e complexos.
b) É uma linguagem de scripting que se destina a ser usada apenas em páginas HTML.
c) Sendo uma linguagem dinâmica, suporta tipificação dinâmica e escopo dinâmico.
d) Uma variável Javascript ganha o seu tipo no momento em que recebe um valor pela primeira vez, e depois o seu tipo nunca muda mais.

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

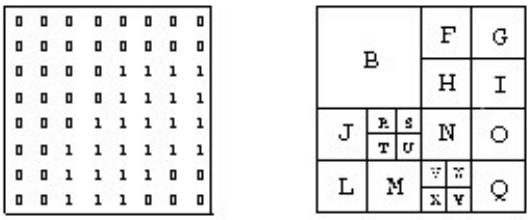
```
let f a b c = a 0 + (if b then c else c)
```

3. [7 valores] **Quadtrees em OCaml**. Uma importante técnica de compressão de imagens bidimensionais baseia-se numa representação usando árvores especiais chamadas **quadtrees**. As quadtrees ajudam a identificar e eliminar redundâncias das imagens, além de permitirem implementar eficientemente operações do tipo união ou intersecção de imagens.

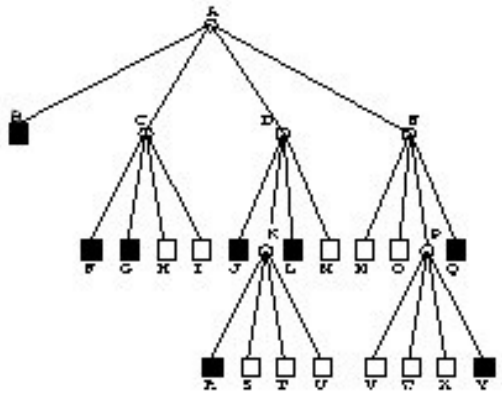
Na base das quadtrees, está a ideia de que qualquer imagem quadrada pode ser decomposta em quatro quadrantes. Cada quadrante pode, por sua vez, ser decomposto em quatro sub-quadrantes e assim sucessivamente, até se atingir o nível do pixel, se necessário. Esta decomposição sucessiva em quadrantes pode ser registada numa árvore quaternária, na qual cada quadrante é representado por meio duma sub-árvore distinta.

Evidentemente que se toda a imagem tiver a mesma cor, então não vale a pena subdividir a imagem: nesse caso a imagem pode ser globalmente representada por um único nó com a cor da imagem. Em geral, um quadrante só precisa de ser subdividido se incluir sub-áreas com cores diferentes.

Para exemplificar, apresenta-se uma imagem a preto e branco, à esquerda, e a respectiva decomposição em quadrantes, à direita. Na imagem, cada dígito 0 representa um pixel preto e cada dígito 1 representa um pixel branco. É a distribuição de cores na imagem que determina a sua decomposição em quadrantes.



A decomposição da direita também pode ser expressa através da quadtree abaixo, a que vamos dar o nome de `example`.



Numa quadtree, convenciona-se que os quatro filhos de cada nó interno estão ordenados pela ordem NW, NE, SW, SE. As folhas da quadtree que correspondam a quadrantes de cor uniforme não precisam de ser subdivididos - é o que acontece no quadrante B, por exemplo.

Em OCaml, o tipo das quadtrees pode definir-se assim:

```
type quadtree = QColor of int | QNode of quadtree *  quadtree * quadtree * q uadtree;;
```

Num termo da forma `QColor x`, só permitimos que x assuma os valores 0 (preto) ou 1 (branco). (No projeto prático, a representação era parecida com esta, mas não era exatamente igual.)

Para dar um exemplo de manipulação de quadtrees, a seguinte função determina o número total de folhas que ocorrem numa quadtree, ou seja o número total de quadrantes com cor uniforme. No caso da árvore `example`, o resultado é 19.

```
let rec count q =
    match q with
        QColor _  -> 1
      | QNode(a,b,c,d)  -> count a + count b + count c + count d
;;
```

Nos problemas que se seguem, trabalha-se sempre com quadtrees devidamente compactadas, i.e. onde os quadrantes de cor uniformes nunca aparecem subdivididos (serem subdivididos seria desnecessário, um desperdício…).

Resolva os problemas que se seguem usando o método indutivo e usando, tanto quanto possível, usando funções simples de tipo 1. Pode definir funções auxiliares, sempre que precisar ou lhe der jeito.

a) [1 valor] Escreva em OCaml uma função para determinar a proporção de branco numa imagem. O resultado é um número real entre 0.0 e 1.0.

```
whiteness: quadtree -> float
 (whiteness example = 0.40625000  [26/64])
 (whiteness (QColor 0) = 0.0)
 (whiteness (QColor 1) = 1.0)
```

b) [2 valores] Escreva em OCaml uma função para produzir uma nova imagem a partir da fusão de duas imagens dadas. Para obter a cotação máxima, a imagem resultante deve ficar compactada. A fusão é baseada numa operação OR que olha para as cores como valores lógicos: o preto (0) funciona como *false* e o branco (1) como *true*.

```
qor: quadtree -> quadtree -> quadtree
 (qor example example = example)
 (qor example (QColor 0) = example)
 (qor example (QColor 1) = QColor 1)
```

Complete o que falta:

```
let rec qor q1 q2 =
    match q1, q2 with
```

c) [2 valores] É possível codificar uma quadtree numa lista de carateres onde aparecem apenas os carateres 'Q' (representa um nó interno), '0' (representa um quadrante preto) e '1' (representa um quadrante branco) Pedimos-lhe para escrever em OCaml uma função que efectue essa conversão. As regras de conversão são as seguintes:

- Na primeira posição da lista coloca-se a raiz da quadtree (será 'Q', '0' ou '1').
- Depois, no caso da quadtree conter sub-árvores (e só nesse caso):
  - Nas posições seguintes da mesma lista coloca-se a primeira sub-árvore da quadtree;
  - Depois coloca-se a segunda sub-árvore da quadtree;
  - Depois coloca-se a terceira sub-árvore da quadtree;
  - Finalmente, a seguir, coloca-se a quarta sub-árvore da quadtree.

```
tolist: quadtree -> char list
 (tolist example = ['Q';'0';'Q';'0';'0';'1';'1';'Q';'0';'Q';'0';'1';'1';
                    '1';'0';'1'; 'Q';'1';'1';'Q';'1';'1';'1';'0';'0'])
 (tolist (Color 0) = ['0'])
```

d) [2 valores] Escreva em OCaml uma função que efetue a conversão inversa da alínea anterior, ou seja que converta listas em quadtrees. Assume-se que a função só será aplicada a listas bem formadas, que realmente codificam quadtrees compactadas verdadeiras. (Os detalhes da solução deste problema têm algumas complicações e este será, em princípio, o problema mais difícil deste grupo.)

```
fromlist: char list -> quadtree
 (fromlist ['Q';'0';'Q';'0';'0';'1';'1';'Q';'0';'Q';'0';'1';'1';
            '1';'0';'1'; 'Q';'1';'1';'Q';'1';'1';'1';'0';'0'] = example)
 (fromlist ['0'] = QColor 0)
```

Nome: Num:

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

```
void F(void) {
  int a = 0;
  void G(void) {
      void H(void) {  a++; F(); }
      H();
  }
  G();
}

int main(void) {
  F();
  return 0;
}
```

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

Use as convenções habituais das aulas: Para efeito 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 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 ;
```

Considere também o seguinte tipo que define **listas duplamente ligadas** e onde cada nó contém dois apontadores, um para o nó sucessor e outro para o nó predecessor:

```
typedef struct DNode {
    Data data ;
    struct DNode *prev, *next ;
} DNode, *DList ;
```

Para criar nós de listas duplamente ligadas, chame a seguinte função, já programada:

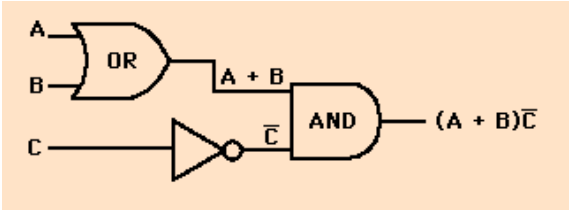
```
DList NewDNode(Data val, DList prev, DList next)
{
    DList n = malloc(sizeof(DNode)) ;
    if( n == NULL )
        return NULL ;
    n->data = val ;
    n->prev = prev ;
    n->next = next ;
    return n ;
}
```

Agora o problema: Escreva uma função chamada `List2DList` que crie, a partir duma lista normal de tipo `List`, uma lista duplamente ligada com os mesmos elementos da lista original e pela mesma ordem.

```
DList List2DList(List l) ;
```

É mais valorizada uma solução iterativa (programada usando ciclos) do que uma solução recursiva. Além disso, quanto mais simples for a sua solução, melhor. Tente criar a lista numa única passagem, preenchendo logo os apontadores `prev` e `next`. Se tiver de fazer mais do que uma passagem pela lista, paciência, mas faça o melhor que conseguir.

6. [5 valores] **Circuitos digitais** são circuitos electrónicos funcionam em lógica binária, o significa que toda a informação é processada usando os valores booleanos ***false*** e ***true***. O objectivo deste problema é a definição em C++ dum sistema de interfaces e classes adequado à representação de circuitos digitais. Vamos considerar circuitos com múltiplas linhas de entradas e apenas uma linha de saída. Assumimos que nos circuitos não existem ciclos (ou seja, não há realimentação de sinais). Eis um exemplo de circuito digital com três linhas de entrada (neste caso, chamadas A, B, e C) e uma linha de saída.



Um circuito é constituído por **elementos** com um número restrito de formas. Cada elemento possui um certo número de linhas de entrada (inputs) e, sempre, uma linha de saída (output). Eis a descrição da forma dos elementos com que vamos trabalhar:

`InLine(string)`	Tem zero inputs, tem um nome, e produz um valor que é consultado num *mapa de entradas* (ver abaixo). Por exemplo, uma linha de entrada chamada "A" é representada por `InLine("A")`.
`Not(a)`	Tem um input e produz o complementar do valor do seu input.
`And(a,b)`	Tem dois inputs e produz a conjunção lógica dos dois valores de input.
`Or(a,b)`	Tem dois inputs e produz a disjunção lógica dos dois valores de input.
`If(a,b,c)`	Tem três inputs e, dependendo do valor de a, produz os valores de b ou c.
`Pool(a,b,c)`	Tem três inputs e produz o valor que ocorre maioritariamente nos inputs.
`False`	Tem zero inputs e produz sempre o valor constante false.
`True`	Tem zero inputs e produz sempre o valor constante true.

O valor de saída dum circuito, calculado usando a função **`Eval`**, depende dos valores que forem escolhidos para as linhas de entrada. Um mapa de entradas indica, para cada nome de linha de entrada, qual o valor booleano a usar em cada entrada. No nosso exemplo, se usarmos um mapa de entradas que associe às três linhas de entrada ("A", "B" e "C") o valor false, então o valor da saída será também false (é fazer as contas…). Use o tipo `map<string,bool>`, da biblioteca do C++, para representar mapas de entrada. Para consultar um mapa m, usa-se assim o operador de indexação, neste exemplo para a entrada A: `m["A"]`.

Problema

Usando um sistema de classes (e uma interface), defina em C++ um tipo soma **Elem** para representar os elementos dum circuito digital. Esse tipo deve suportar as duas operações públicas indicadas a seguir. Você vai ter de decidir quais são as classes abstractas e as classes concretas a definir. **É fundamental escrever código bem fatorizado e extensível, pois em grande medida é isso mesmo que esta pergunta pretende avaliar**.

```
int Count(const string &s)-Conta quantas vezes a linha de entrada s ocorre no circuito.

bool Eval(const map<string,bool> &m) - Calcula o valor de saída dum circuito digital. O
mapa de entradas m indica quais os valores a considerar para as linhas de entrada.
Assuma que o mapa não tem erros, ou seja que o mapa contém valores para todas as linhas
de entrada usadas.
```

Na sua resolução, mostre primeiro o conteúdo do ficheiro `Elem.h`. Depois mostre o conteúdo do ficheiro `Elem.cpp`.

Para poupar tempo não escreva os construtores nem destrutores, nem mesmo os seus cabeçalhos. Implemente apenas as variáveis de instância nas classes apropriadas e as duas funções pedidas nas classes apropriadas. Também não faça includes, nem trate da burocracia associada aos includes.

NOTA1: Repare que este programa obriga a escrever um grande número de classes, mas felizmente todas elas são muito pequenas, além de que os construtores e destrutores não são para escrever.

NOTA2: Este problema tem algumas semelhanças com o problema das expressões algébricas da aula teórica 20.