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

07 de Julho 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) Ao contrário da linguagem Java, nas linguagens C e C++ não é efetuada a validação de índices nos acessos a arrays. Este facto proporciona ao C e ao C++ o quê?

a) Maior facilidade em aprender a linguagem.
b) Maior eficiência.
c) Maior segurança.
d) Maior legibilidade dos programas.

B) No C++, as declarações dos membros privados das classes aparecem no ficheiro ".h", e portanto podem ser obser+vados pelo programador que escreve código cliente dessas classes. Até que ponto é que isso é um problema?

a) É um problema pois significa que o C++ não suporta encapsulamento ao nível das classes. Essa é uma das razões pela qual a linguagem C++ é insegura.
b) Não é um problema porque, apesar de tudo, o programador não pode escrever código que dependa desses detalhes privados (apesar de os poder observar).
c) Não é um problema porque o programador usa autodisciplina e evita aceder aos membros privados.
d) Não é um problema porque o programador pode escolher nomes "esquisitos" para os métodos privados, nomes que ninguém entende e que ocultam o propósito desses membros privados.

C) No seguinte programa em C, qual o âmbito da variável global `a`?

```
int a ;
void f(int a) { printf("%d\n", a) ; }
void g(int b) { f(a+b); }
int main(void) { g(5); exit(0); }
```

a) O programa completo.
b) O interior da função f.
c) Todo o programa, excepto o interior da função f.
d) O âmbito é vazio.

D) O *static link* do registo de ativação duma função F aponta para:

a) O topo da pilha de execução.
b) A árvore sintática de F.
c) O registo de ativação da função que chamou F.
d) O registo de ativação da função dentro da qual F está definida.

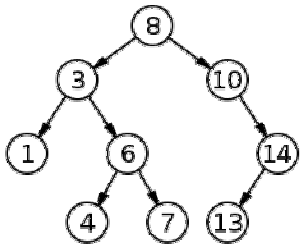
E) O que será mais complicado, no caso geral: (1) tomar um programa C++ e reescrevê-lo manualmente em Javascript (JS); ou (2) tomar um programa JS e reescrevê-lo manualmente em C++?

a) O JS é mais liberal do que o C++ a todos os níveis, desde o sistema de tipos dinâmico até à existência de operações de modificação direta da estrutura dos objetos. Por isso, é mais trabalhoso converter de C++ para JS.
b) O JS é mais liberal do que o C++ a todos os níveis, desde o sistema de tipos dinâmico até à existência de operações de modificação direta da estrutura dos objetos. Por isso, é mais trabalhoso converter de JS para C++.
c) Escrever o programa original é que foi complicado, pois a atividade da programação é exigente. Não há diferença significativa de esforço em traduzir em qualquer das duas direções.
d) O esforço de conversão será idêntico, mas só se envolver uma terceira linguagem (intermédia) de muito alto nível (como o OCaml) e se fizer sempre uma tradução suplementar através dessaa linguagem intermédia.

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

```
let f a b c = if a b then a c else c
```

3. [7 valores] **Árvores binárias de pesquisa OCaml**. Uma árvore binária de pesquisa (*binary search tree* - BST) é uma árvore binária com as seguintes propriedades:



- Todos os valores contidos na subárvore esquerda são menores do que o valor da raiz.
- Todos os valores contidos na subárvore direita são maiores do que o valor da raiz.
- Tanto a subárvore esquerda como a subárvore direita, também são BSTs.

Repare que todos os valores são distintos numa BST. A árvore vazia também é uma BST.

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

```
type 'a tree = Nil | Node of 'a * 'a tree * 'a tree ;;
```

Para dar um exemplo de como determinadas operações ficam muito eficientes nas BSTs, a seguinte função determina o menor valor que ocorre numa BST. Como é natural, a função só está definida para BSTs não vazias.

```
let rec bmin q =
    match q with
      | Node(x,Nil,_) -> x       (* árvore esquerda vazia -> x já é o mais pequeno *)
      | Node(_,l,_) -> bmin l    (* árvore esquerda não vazia -> procurar nela *)
;;
```

Resolva os problemas que se seguem usando o método indutivo. Pode definir funções auxiliares, sempre que precisar ou lhe der jeito. **É proibido converter as árvores para listas; resolva todos os problemas usando árvores diretamente.**

a) [1.5 valor] Escreva em OCaml uma *função eficiente* para verificar se um dado valor ocorre numa BST.

```
bsearch: 'a -> 'a tree -> bool
```

b) [1.5 valor] Escreva em OCaml uma *função eficiente* para inserir um valor numa BST. O resultado deve ser uma BST. Se o valor já estiver na árvore, não é inserido.

```
binsert: 'a -> 'a tree -> 'a tree
```

c) [1.5 valor] Escreva em OCaml uma *função eficiente* para remover o elemento mais pequeno duma BST. Se a BST for vazia, então é retornada a BST vazia.

bdelmin: `'a tree -> 'a tree`

d) [1.5 valor] Escreva em OCaml uma *função eficiente* para remover um valor duma BST. Se o valor não ocorrer, então é retornada uma BST igual à original. [Ajuda: o ramo mais complicado desta função pode ser programado com ajuda das funções `bmin` e `bdelmin`.]

bdel: `'a -> 'a tree -> 'a tree`

e) [1 valor] Escreva uma *função eficiente* para verificar se uma árvore binária é uma BST. Repare que se usar diretamente a definição de BST e o método indutivo obtém uma função com complexidade exponencial. A solução exponencial não nos interessa e vale 0 valores; também não pode converter a árvore para lista. Há diversas formas de resolver este problema.

bcheck: `'a tree -> bool`

Nome: Num:

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

```
void F(int a) {
  void G(void) { a++ ; Ψ }
  if( a <= 1 )
    G();
  else
    F(a-1);
}

int main(void) {
  F(3);
  return 0;
}
```

Mostre qual o estado da pilha de execução no momento em que a execução do programa atinge o ponto marcado com a letra grega Ψ (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.

Só é preciso usar parte das células de memória apresentadas abaixo.

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.

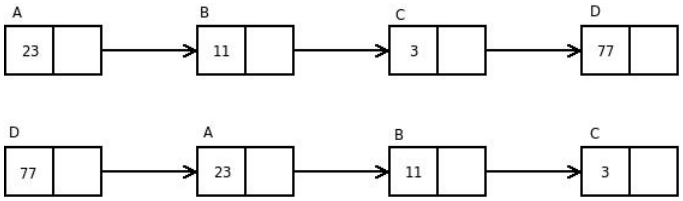
```
typedef double Data ;
typedef struct Node {
    Data data ;
    struct Node *next ;
} Node, *List ;
```

Escreva uma função chamada `Rotate` que transfira o último nó duma lista para a primeira posição. Não é criado qualquer nó novo, sendo a lista do resultado constituída pelos mesmos nós da lista original, só que por uma ordem diferente. (Não basta transferir valores; é mesmo preciso mudar a posição dos nós na lista).

O cabeçalho da função é o seguinte:

```
List Rotate(List l);
```

Exemplo: Aplicando a função `Rotate` à primeira lista, deve obter-se a segunda lista.



Procure uma solução simples e compacta.

6. [5 valores] **Sistema de ficheiros**. Um sistema de ficheiros consiste numa árvore n-ária na qual podem ocorrer os três tipos de **componentes**:

- **File** – Caracteriza-se por um *nome* (uma string) e por um *texto* representado por um vetor de linhas (vetor de strings).
- **Directory** – Caracteriza-se por um *nome* (uma string) e por um *conteúdo* representado por um vetor de componentes (ficheiros, diretorias e links). O conteúdo está ordenado crescentemente pelos nomes das componentes.
- **Link** – Caracteriza-se por um *nome* (uma string) e por um *caminho absoluto* (ver definição mais abaixo) que identifica outra componente do mesmo sistema. Os links também são conhecidos como "shortcuts", "atalhos" ou "aliases". Se o caminho absoluto guardado num link não identificar qualquer componente existente do sistema, diz-se que o link está **quebrado** (é uma situação que ocorre nos sistemas de ficheiros normais). Um link quebrado nunca pode ser percorrido. Permite-se que um link aponte para outro link.

Chama-se **caminho absoluto** duma componente, à sequência de nomes que assinala um caminho descendente desde a raiz do sistema até essa componente (por exemplo, a sequência: "root", "home", "jose", "tmp2"). Repare que qualquer componente fica completamente identificada dentro dum sistema de ficheiros através do seu caminho absoluto. Um caminho absoluto nunca é vazio, pois contém sempre o nome da raiz. Um caminho absoluto é representado usando um vetor de strings.

Problema

O objectivo deste problema é definir um sistema de interfaces e classes adequado a uma representação de sistemas de ficheiros. Atendendo ao que se prevê relativamente à evolução futura do programa, é suficiente definir uma interface (chamada **FileSys**), uma classe abstracta (chamada **FileSysAbs**), e três classes concretas (chamadas **File**, **Directory** e **Link**). Mas repare que para a sua solução ter qualidade, vai ser fundamental **factorizar o máximo de código** na classe abstracta.

Descrevem-se a seguir as operações públicas a definir. Na maioria das situações, estas funções são para aplicar à raiz dum sistema de ficheiro.

`int CountFiles() const` – Conta o número de componentes de tipo File que existem num sistema de ficheiros.

`bool Validate() const` – Verifica se um sistema de ficheiros é válido, ou seja: [1] se cada componente tem um nome não vazio; [2] se o conteúdo de cada directoria está ordenado por nome; [3] se cada link tem um caminho não vazio. Não há mais nada para validar.

`FileSys *Find(vector<string> path) const` – Pesquisa num sistema de ficheiros qual a componente de tipo File ou Directory que é identificada por um caminho dado. Se o caminho não identificar nada, então a função retorna 0 (Null). Sempre que se atinge um link, a pesquisa continua na componente apontada por ele. [Se você não processar links nesta função, só perde 40% da cotação da função.]

O seu programa deve ter apenas dois ficheiros: `FileSys.h` e `FileSys.cpp`. Mostre primeiro o conteúdo do ficheiro `FileSys.h` e depois o conteúdo do ficheiro FileSys.cpp.

Para ganhar tempo não escreva os construtores nem os destrutores. Também não faça includes nem trate da burocracia associada a includes.

Neste problema é preciso usar os tipos de biblioteca `std::string` e `std::vector`. Nas aulas teóricas 18 e 19 são mostrados exemplos de utilização destes tipos.

Pode definir funções auxiliares sempre que precisar.

Resolva este problema usando a frente e o verso das folhas.