Licenciatura em Engenharia Informática (FCT/UNL)
Ano Lectivo 2011/2012
Linguagens e Ambientes Programação – Época Normal

20 de Junho de 2012 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. Responda no próprio enunciado, usando a frente e o verso.*
Nos problemas em OCaml mostre que sabe usar o método indutivo.
Pode definir funções/métodos auxiliares sempre que precisar (muitas vezes é mesmo preciso).
Pode usar funções/métodos de biblioteca, como por exemplo do módulo List do OCaml.
Normalmente, respostas imperfeitas merecem alguma pontuação.
Fraude implica reprovação na cadeira.

1. [2 valores] Escolha múltipla (as respostas erradas não descontam). Indique as respostas mais correctas aqui:

A	B	C	D	E

A) Nas implementações habituais dos compiladores da linguagem C (gcc, por exemplo), o endereço das variáveis locais das funções é determinado:

a) Completamente em tempo de compilação.
b) Completamente em tempo de execução.
c) Parcialmente em tempo de compilação e parcialmente em tempo de carregamento.
d) Parcialmente em tempo de compilação e parcialmente em tempo de execução.

B) Sobre escopo estático e escopo dinâmico:

a) A linguagem JavaScript suporta apenas escopo estático.
b) A linguagem C suporta apenas escopo estático.
c) A linguagem Java suporta apenas escopo dinâmico.
d) A linguagem OCaml suporta apenas escopo dinâmico.

C) Numa função em C++, dois parâmetros podem referir a mesma zona de memória?

a) Sim. Por exemplo, se os dois forem passados por valor.
b) Sim. Por exemplo, se um for passado por valor e o outro por referência constante.
c) Sim. Por exemplo, se um for passado por referência constante e o outro por referência normal.
d) Nunca.

D) Que linguagens suportam inferência de tipos?

a) Java e C.
b) Java e OCaml.
c) JavaScript e C.
d) C e C++.

E) Suponha que um programa em Java foi compilado e não gerou qualquer erro ou warning?

a) É garantido que programa vai funcionar de acordo com o pretendido.
b) Se durante a execução não ocorrer qualquer ciclo infinito, o resultado final estará correto.
c) Se durante a execução não ocorrer qualquer ciclo infinito, o programa terminará normalmente produzindo um resultado, mesmo que seja um resultado incorreto.
d) Nenhuma das anteriores.

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

```
void H(int *pt) {
    *pt = 456;
    Ψ
}
void F(int *pt) {
    void G(int *pt) { H(pt+1); }
    G(pt+1);
}

int main(void) {
    int v[] = {100, 101, 102} ;
    F(v);
    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`).

Repare que a variável local `v` ocupa três células consecutivas na zona das variáveis locais da função `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.

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

Nome: Num:

3. [7 valores] Considere a seguinte representação para sistemas de ficheiros hierárquicos:

Um **sistema de ficheiros hierárquico** consiste numa árvore n-ária cujos nós terminais se chamam **ficheiros** e cujos nós internos se chamam **diretorias**. Um **ficheiro** caracteriza-se por um nome e por uma lista de linhas de texto. Uma **diretoria** caracteriza-se por um nome e por uma lista de ficheiros e diretorias.

No contexto dum sistema de ficheiros, define-se **caminho absoluto** dum ficheiro ou diretoria como sendo a lista de nomes que assinalam o caminho desde a raiz do sistema até ao ficheiro ou diretoria em causa. Note que qualquer elemento dum sistema é identificado de forma perfeitamente clara pelo seu caminho absoluto. O nome da raiz do sistema não faz parte dos caminhos absolutos pois a localização da raiz nunca oferece dúvida (e, portanto, por exemplo, o caminho absoluto da raiz é a lista vazia).

Nos nossos sistemas de ficheiros, vamos ainda considerar **links**, por vezes designados por *shortcuts* ou *aliases*. Um **link** caracteriza-se por um nome e por um caminho absoluto e permite designar outra componente do sistema que seja identificada por esse caminho absoluto.

Eis o tipo dos sistemas de ficheiros hierárquicos em OCaml:.

```
type fileSys =
          File of string * string list    (* nome + texto *)
        | Dir of string * fileSys list    (* nome + lista de fich e dirs *)
        | Link of string * absPath ;;     (* nome + caminho absoluto *) ;;
```

Eis o tipo dos caminhos absolutos:

```
type absPath = string list ;;
```

Eis um exemplo dum sistema de ficheiros constituído por 2 diretorias, 3 ficheiros e 5 links:

```
let fs = Dir("root",
           [ Dir("dd",[File("z",["zzzzzz"]); Link("l",["c"])]);
             File("f",["Era uma vez"]);
             Link("lp",["z"]);
             Link("z",["p"]);
             Link("erro",["e";"r"]);
             File("p",["hello"]);
             Link("c",["dd";"l"])
         ]) ;;
```

No exemplo anterior, há diversos links estranhos. Repare que o link "**erro**" é **inválido** pois contém um caminho absoluto inexistente. Repare também que os links "**l**" e "**c**", apesar de serem válidos, se referem mutuamente, constituindo um **ciclo**. Um **ciclo** é uma cadeia circular de links.

Uma operação importante sobre sistemas de ficheiros é a que permite obter o elemento que é identificado por um caminho absoluto. A função `get`, que lhe mostramos abaixo, implementa esta operação. Repare que a função gera uma exceção no caso do caminho absoluto ser **inválido**.

```
let getName z =
    match z with
      File(name, _) -> name | Dir(name, _) -> name | Link(name, _) -> name ;;

let rec get path z =
    match path, z with
      [],z -> z
    | x::xs, Dir(name, z::zs) ->
        if x = getName z then get xs z
        else get path (Dir(name, zs))
    | _, _ -> raise (Arg.Bad "get: invalid path") ;;
```

Escreva em OCaml, usando o método indutivo (tanto quanto possível), as cinco funções sobre sistemas de ficheiros que se encontram nas próximas duas páginas.

a) [1 valor] A função countLinks conta o número de links existentes num sistema de ficheiros.

```
countLinks : fileSys -> int
   countLinks fs = 5
```

b) [1 valor] A função `validPath` testa se um caminho absoluto é válido num sistema, ou seja se refere algum ficheiro, diretoria ou link. [Ajuda: há soluções simples para este problema que usam a função `get` como auxiliar.]

```
validPath : absPath -> fileSys -> bool
  validPath ["erro"] fs = true
  validPath ["dd"] fs = true
  validPath ["dd"; "l"] fs = true
  validPath ["bla"; "bla"; "bla"] fs = false
```

c) [1.5 valor] A função `get2` faz o mesmo que a função `get`, só que tem o poder adicional de *resolver* links. **Resolver um link** significa descobrir ou ficheiro ou diretoria que esse link refere duma forma direta ou indireta. Resolver um link pode envolver navegar numa sequência de links interligados. O resultado de `get2` nunca é um link mas sim um ficheiro ou uma diretoria. Estude bem os exemplos abaixo para perceber o que a função `get2` deve fazer. [Ajuda: usando a função get como auxiliar, conseguirá escrever uma função bastante simples.]

```
get2 : absPath -> fileSys -> fileSys
  get2 ["p"] fs = File("p",["hello"])
  get2 ["lp"] fs = File("p",["hello"])
  get2 ["erro"] fs = (* gera exceção porque o link é inválido *)
  get2 ["c"] fs = (* a avaliação não termina por causa do ciclo *)
```

d) [2 valor] A função `badLinks` produz a lista de todos os links inválidos dum sistema de ficheiros. [Ajuda: para nunca perder acesso ao sistema de ficheiros completo, poderá passar o sistema de ficheiros duas vezes (ou seja, repetido) para uma função auxiliar.]

```
badLinks : fileSys -> fileSys list
  badLinks fs = [Link("erro",["e";"r"])]
```

e) [1.5 valores] A função `hasCycle` testa se um sistema de ficheiros contém algum ciclo. [Ajuda: poderá usar numa função auxiliar um argumento suplementar de tipo "lista de caminhos absolutos" para detetar que já passou pelo mesmo sítio duas vezes durante a tentativa de resolução de cada link.]

```
hasCycle : fileSys -> bool
  hasCycle fs = true
```

Nome: Num:

4. [2 valores] Diga qual o tipo da seguinte função em OCaml:

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

5. [3 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 int Data ;
typedef struct Node {
    Data data ;
    struct Node *next ;
} Node, *List ;
```

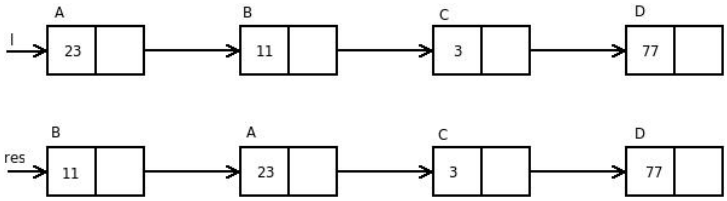
Escreva uma função chamada `swap` que, dada uma lista `l` e um inteiro `idx`, troque o nó que se encontra na posição `idx` com o nó que se encontra na posição `idx+1`. A lista do resultado deve ser constituída pelos mesmos nós da lista original, só que por uma ordem diferente. (Não basta transferir valores; é mesmo preciso mudar a posição dos nós na lista).

A função `swap` só faz mudanças caso existiam nós nas duas posições `idx` e `idx+1`. Se não existirem, a lista original é retornada.

O cabeçalho da função é o seguinte:

```
List swap(List l, int idx)
```

Por exemplo, a chamada `swap(l,0)`, deve converter a primeira lista na segunda lista.



Procure uma solução simples e compacta, que não use recursividade e que percorra a lista apenas uma vez.

[Ajuda: há diversas técnicas para escrever esta função, mas a que talvez permita tratamento mais uniforme dos vários casos envolve começar por adicionar um nó inicial suplementar que depois é eliminado no final.]

6. [4 valores] Num jogo de computador escrito em C++ existem três tipos de personagens: Coelho, Caçador, Cenoura.

O **Coelho** tem por objectivo sobreviver e comer o máximo de Cenouras. É caracterizado por uma posição (x, y), uma velocidade, e pelo número de cenouras comidas até ao momento.

O **Caçador** tem por objectivo proteger as cenouras, mesmo que para isso tenha de matar o Coelho. É caracterizado por uma posição (x, y), por uma velocidade, e pelo número de balas que lhe restam na arma.

As **Cenouras** são passivas. Caracterizam-se por uma posição (x, y), e podemos considerar que têm uma velocidade zero.

O objectivo deste problema é a definição em C++ dum sistema de interfaces e classes **bem fatorizado e extensível** adequado à representação destas personagens. O sistema tem de estar preparado para a futura introdução de novas personagens.

Problema

Defina interfaces, classes abstratas e classes concretas que permitam representar as personagens, com as suas variáveis de instância e os seus construtores. Não defina mais nenhuma função para além dos construtores. As classes concretas devem chamar-se **Rabbit**, **Hunter** e **Carrot**.

Apesar deste ser um problema limitado, é necessário tomar diversas decisões para garantir que o sistema fica bem fatorizado e depois exprimir essas decisões usando código C++ correto.