Mestrado Integrado em Engenharia Informática (FCT/UNL)
Ano Letivo de 2018/2019
Linguagens e Ambientes Programação – Solução Teste 1

16 de abril de 2019 às 18:30

Teste com consulta com 1 hora e 30 minutos de duração

Nome: Num:

Notas: *Este enunciado é constituído por 3 grupos de perguntas. Responda no próprio enunciado,* ***usando a frente e o verso****.*
Normalmente, respostas imperfeitas merecem alguma pontuação.
Fraude implica reprovação na cadeira.

1. [3 valores] Escolha múltipla. As respostas erradas não descontam. **Indique as respostas nesta tabela:**

A	**B**	**C**
d	b	c

A) Em função do momento de ligação, como é que podemos caracterizar a invocação de métodos em Java? (A pergunta-se aplica-se a métodos de instância e não a métodos de classe.)

a) Tratam-se de ligações puramente estáticas, em todas as situações.

b) Tratam-se de ligações semidinâmicas, em todas as situações.

c) Tratam-se de ligações puramente dinâmicas, em todas as situações.

d) Geralmente tratam-se de ligações semidinâmicas, mas no caso particular de métodos privados, as ligações são estáticas pois métodos privados não podem ser redefinidos nas subclasses.

B) Sobre a linguagem C. Qual a alternativa correta?

a) Não existe padronização oficial, existindo contudo uma norma *de facto* baseada num famoso livro publicado em 1978.

b) A linguagem tem vindo a evoluir e existem algumas padronizações publicadas por organismos oficiais de normalização.

c) Existe uma padronização oficial, mas tem mais do que uma década porque a linguagem C parou de evoluir.

d) A primeira norma oficial do C é mais recente do que a última norma oficial do OCaml.

C) Para executar um programa, temos geralmente duas alternativas: (1) usar diretamente um interpretador; (2) começar por submeter o programa a um compilador, antes de executar. Qual das seguintes hipóteses é a correta?

a) Os interpretadores executam o código fonte diretamente, sem qualquer tradução prévia. Não há exceções.

b) Os compiladores executam o código fonte diretamente, sem que tenha lugar qualquer processo de tradução.

c) Interpretação e compilação não são mutuamente exclusivos e muitos interpretadores efetuam algum trabalho de tradução inicial, semelhante ao dos compiladores.

d) Uma máquina virtual, como a JVM, é um exemplo dum compilador puro.

2. [3 valores] Diga qual o tipo da seguinte função em OCaml. A função chama-se **`f`** e tem dois argumentos.

```
let rec f (x::xs) (y::ys) = x::(f xs (y::ys)) ;;
```

Solução:

```
'a list -> 'b list -> 'a list
```

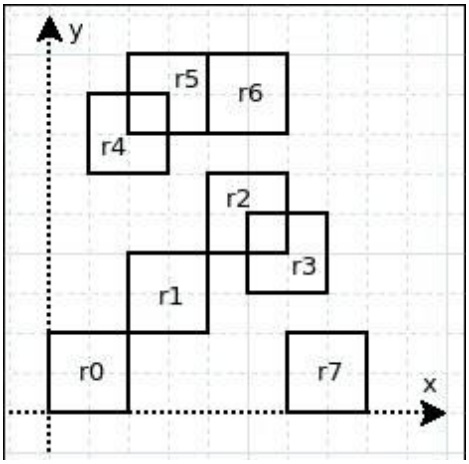
3. Coleções de retângulos - Vamos trabalhar com retângulos alinhados com os eixos do plano cartesiano. Eis o tipo OCaml dos nossos retângulos:

```
type rect = { left: float; bottom: float; right: float; top: float  }
```

Repare que conseguimos indicar os limites de cada retângulo usando apenas quatro coordenadas. Por exemplo, o seguinte retângulo

```
let r0 = {left = 0.0; bottom = 0.0; right = 2.0; top =  2.0}
```

é um quadrado que tem a base esquerda na origem `(0.0,0.0)` e o topo direito no ponto `(2.0,2.0)`.



Para representar coleções de retângulos usamos listas. Eis a coleção de retângulos da imagem ao lado - por acaso são todos quadrados, mas podiam não ser:

```
let example = [
      (*r0*) {left = 0.0; bottom = 0.0; right = 2.0; top = 2.0};
      (*r1*) {left = 2.0; bottom = 2.0; right = 4.0; top = 4.0};
      (*r2*) {left = 4.0; bottom = 4.0; right = 6.0; top = 6.0};
      (*r3*) {left = 5.0; bottom = 3.0; right = 7.0; top = 5.0};
      (*r4*) {left = 1.0; bottom = 6.0; right = 3.0; top = 8.0};
      (*r5*) {left = 2.0; bottom = 7.0; right = 4.0; top = 9.0};
      (*r6*) {left = 4.0; bottom = 7.0; right = 6.0; top = 9.0};
      (*r7*) {left = 6.0; bottom = 0.0; right = 8.0; top = 2.0} ]
```

Oferecemos-lhe, já programada, esta função booleana (pouco óbvia) que testa se dois retângulos se intersetam:

```
let intersect a b =
     b.left <= a.right && a.left <= b.right && b.bottom <= a.top && a.bottom <=  b.top
```

Dois retângulos intersetam-se se tiverem pelo menos um ponto em comum.

Nos problemas que se seguem, mostre que sabe usar o método indutivo e escreva, se possível, funções de categoria 1 ou 2. Pode também usar funções auxiliares escritas por si ou pertencentes à biblioteca padrão do OCaml. Não use mecanismos ou raciocínios imperativos nem simule mecanismos ou raciocínios imperativos. Para resolver cada problema, pode chamar funções das alíneas anteriores, mesmo que não tenha resolvido essas alíneas. Não é importante a ordem dos resultados das funções. Assuma que as funções são sempre aplicadas a argumentos bem formados.

a) [3 valores] Escreva uma função **`quadrant1`** que, dada uma coleção, determine quantos retângulos se situam integralmente no 1º quadrante do plano cartesiano. O 1º quadrante é definido pela condição **`x>=0 && y>=0`**.

```
quadrant1: rect list -> int
```

Exemplos: `quadrant1 example = 8` `quadrant1 [] = 0`

Solução:

```
let rec quadrant1 l =
    match l with
       [] -> 0
     | r::rs -> (if r.bottom >= 0.0 && r.left >= 0.0 then 1 else 0) + quadrant1 rs
;;
```

b) [3 valores] Escreva uma função **`neighbours`** para obter, numa coleção, todos os retângulos vizinhos dum retângulo dado, i.e. todos os rectângulos que intersectam o retângulo dado. Contudo, o retângulo dado não é considerado vizinho de si próprio.

```
neighbours: rect -> rect lixt -> rect lixt
```

Exemplos: `neighbours r2 example = [r1;r3]` `neighbours r7 example = []` `neighbours r1 [] = []`

Solução:

```
let rec neighbours r l =
    List.filter (fun r2 -> r2 <> r && intersect r r2) l
;;

let rec neighbours r l = (* outra maneira *)
    match l with
       [] -> []
     | r2::rs ->
              if r2 <> r && intersect r r2 then
                    r2::neighbours r rs
              else
                    neighbours r rs
;;
```

c) [3 valores] Escreva uma função **solitary** para obter todos os retângulos sem vizinhos duma coleção.

solitary: rect list -> rect list

Exemplos: **solitary** example = [r7] **solitary** [] = []

Solução:

```
let rec solitary l =
    List.filter (fun r -> neighbours r l = []) l
;;
```

d)* [2.5 valores] Escreva uma função **reachable** para determinar, numa coleção, todos os retângulos alcançáveis a partir dum retângulo dado, incluindo o próprio retângulo dado.

reachable: rect -> rect list -> rect list

Exemplos: **reachable** r0 example = [r0;r1;r2;r3] **reachable** r6 example = [r4;r5;r6]
reachable r7 example = [r7]

Solução:

```
let diff l1 l2 =
    List.filter (fun x -> not (List.mem x l2)) l1
;;

let rec reachable r l =
    let n = neighbours r l in
        let rest = diff l (r::n) in
            r :: List.flatten (List.map (fun b -> reachable b rest) n)
;;

let rec reachable r l = (* outra maneira *)
    match l with
        [] -> [r]
      | x::xs -> (if intersect x r then reachable x xs else []) @ reachable r xs
;;
```

e) [2.5 valores] Escreva uma função **islands** para determinar todos os grupos maximais de retângulos, onde os retângulos de cada grupo têm a propriedade de serem mutuamente alcançáveis.

islands: rect list -> rect list list

Exemplos: **islands** example = [[r0;r1;r2;r3]; [r4;r5;r6]; [r7]] **islands** [] = []

Solução:

```
let rec add v ll =
    match ll with
        [] -> [[v]]
      | l::ls -> if neighbours v l <> [] then (v::l)::ls else l::add v ls
;;

let rec islands l =
    match l with
        [] -> []
      | x::xs -> add x (islands xs)
;;
```