

Teste 2022

Resolva:

- Dois dos exercícios 1 a 3, à sua escolha ($2 \times 8 = 16$ valores)
- O exercício 4 (4 valores)

1. Entregue as suas propostas de solução em papel
2. Teste as suas propostas em casa (copie tudo o que fizer) e envie por email um pequeno relatório comentando o que teria de ser alterado no que entregou, por forma a se obter implementações verificadas das funções pedidas.

Exercício 1

Segundo a Wikipedia, o algoritmo de Euclides para o cálculo do *máximo divisor comum* pode ser escrito em pseudo-código como se segue:

```
1 function gcd(a, b)
2   while a ≠ b
3     if (a > b) a := a - b;
4     else b := b - a;
5   return a;
```

Escreva esta função em WhyML, incluindo a sua especificação. Note que poderá utilizar na especificação a função da biblioteca do Why3 `function gcd int int : int`.

```
1 module EuclideanAlgorithm
2
3   use import int.Int
4   use import int.EuclideanDivision
5   use import ref.Refint
6   use import number.Gcd
```

1.

let rec euclid ($u\ v$: int) : int
requires { $u > 0 \wedge v > 0$ }
ensures { result = gcd $u\ v$ }
ensures { result $\leq u$ }
ensures { result $\leq v$ }
ensures { result > 0 }
= if $u < v$ then u
else
if $u > v$ then (euclid (~~$u-v$~~) v)
else (euclid u ($v-u$))

2.

let ~~rec~~ function true-to-list (t : true int) : list int
requires { sorted BT t }
ensures { sorted result }
ensures { forall x : int. occurrences BT $t\ x$ = occurrences result x }
ensures { size BT t = size result }

4.

let ~~rec~~ most-frequent (a : array int) : int
requires { length $a > 0$ }
requires { forall $i\ j$: int. $0 \leq i < j < (\text{length } a) \rightarrow a[i] \leq a[j]$ }
ensures { forall x : int. numof $a\ x\ 0 (\text{length } a) \leq \text{numof } a\ \text{result } 0 (\text{length } a)$ }
ensures { numof $a\ \text{result } 0 (\text{length } a) > 0$ }

Nota: invariante escrito no "codeblock" do enunciado do ex. 4.

```

8   let euclid (u v: int) : int
9       requires { ... }
10      ensures { ... }
11      =
12      ...
13

```

Nota: se preferir apresentar uma implementação recursiva poderá fazê-lo, devendo nesse caso alterar `let` para `let rec`.

Exercício 2

Considere o seguinte tipo polimórfico de árvores binárias:

```

type tree 'a = Empty | Node (tree 'a) 'a (tree 'a)

```

Usando predicados referidos no exercício sobre árvores, escreva um contrato para a seguinte função que converte árvores ordenadas em listas ordenadas, preservando o número de ocorrências dos elementos.

```

let function tree_to_list (t:tree int) : list int

```

Exercício 3

Complete a definição da função seguinte, que conta o número de ocorrências de um elemento x num array u , entre os índices k e $l-1$. Note que não é possível descrever logicamente, através de uma pós-condição, a contagem feita pela função; na verdade a ideia desta definição pura é ser utilizada ao nível lógico no exercício seguinte, para especificar o comportamento de uma outra.

```

1 let rec function numof (u:array int) (x:int) (k:int) (l:int)
2     requires { ... }
3     ensures { 0<=result<=l-k }

```

variant { ... }
= ...

Exercício 4

O programa imperativo em baixo determina o elemento que ocorre com maior frequência num array ordenado. Trata-se de um problema nada trivial no caso geral, mas que no caso de o array estar ordenado tem uma solução de tempo linear óbvia.

Escreva o contrato correspondente a esta especificação informal e todas as restantes anotações necessárias para provar a correção do programa face a esse contrato. Deverá utilizar para isso a função `numof` do ex. anterior.

```
1 use int.Int
2 use ref.Refint
3 use array.Array
4 use array.NumOfEq
5
6 let most_frequent (a: array int) : int
7 = let ref r = a[0] in
8   let ref c = 1 in
9   let ref m = 1 in
10  for i = 1 to length a - 1 do
11    if a[i] = a[i-1] then begin
12      incr c;
13      if c > m then begin m <- c; r <- a[i] end
14    end else
15      c <- 1
16  done;
17  r
```

(invariant { (forall j: int. 0 ≤ j < i → (numof a a[j] ≤ m
∧ !m = numof a !∃ 0 i
∧ !c ≤ !m)) ∧ !m ≤ length a }