Faculdade de Computação
Programação Funcional (BCC/BSI) - 1° Período
**1a. Lista de Exercícios: Expressões e Funções em Linguagem Haskell**

1) Teste as seguintes expressões no sistema WinHugs, descreva a operação realizada e informe o resultado obtido:

---
**(testes no WinHugs)**
---

2) Analise a função seguinte escrita em Haskell e explique sua finalidade.

```
fun m n p = (m==n) && (n==p)
```

**RESPOSTA:**
---
A função *fun* recebe três valores (m, n e p) e retorna verdadeiro (*True*) se os três valores forem iguais, e falso (*False*) se forem diferentes. Alguns exemplos de aplicação são:

```
Hugs> fun 3 2 1 where fun m n p = (m==n) && (n==p)
False

Hugs> fun 1 1 1 where fun m n p = (m==n) && (n==p)
True
```
---

3) Sejam as duas funções abaixo que verificam se um dado número é par. Teste cada função e explique a estratégia utilizada na implementação de cada uma.

```
par x = (mod x 2) == 0

par1 x = if (x == 0) then True
                     else not (par1 (x-1))
```

**RESPOSTA:**
---
Ambas as funções recebem um número e retornam verdadeiro (True) se o mesmo for par. A função *par* utiliza como estratégia obter o resto da divisão do número por dois, e caso seja 0 (zero), o número é par. A função *par1* utiliza uma definição recursiva, em que o caso base é a admissão que 0 (zero) é par. A partir daí, um número é par se o seu anterior for ímpar (não-par) e assim sucessivamente até que 0 seja atingido. Como zero é par, o resultado será verdadeiro para os pares e falso para ímpares.
Exemplo:

```
>par1 3
 not (par1 2)
 not (not (par1 1))
 not (not (not (par1 0)))

 not (not (not True))
```

```
not (not False)
not True
False
```

---

4) Considere a seguinte função escrita em Haskell:

```
test n = if (n `mod` 2 == 0)  then n
                              else test(2 * n + 1)
```

Para quais valores de entrada (n) a função não se encerra? Por que? Use exemplos simples para explicar sua resposta.

**RESPOSTA:**

---

A função não se encerra quando aplicada à um número ímpar, pois a condição (n `mod` 2 == 0) será falsa, e a chamada recursiva sempre se aplica à outro número ímpar (2 * n + 1). Exemplo de testes:

```
Hugs> test 4 where test n = if (n `mod` 2 == 0)  then n
                                                 else test(2 * n + 1)
4

Hugs> test 5 where test n = if (n `mod` 2 == 0)  then n
                                                 else test(2 * n + 1)
{Interrupted!}
```

---

5) Escreva funções para calcular:

(a) Uma equação do primeiro grau (ax + b )
(b) Uma equação do segundo grau ($ax^2 + bx + c$)

**RESPOSTA:**

---

```
primeiro_grau::Float->Float->Float
primeiro_grau x y =  - y / x


delta::Float->Float->Float->Float
delta a b c = -b^2 * 4 * a * c

segundo_grau::Float->Float->Float->(Float, Float)
segundo_grau a b c = if (delta a b c) < 0.0 then undefined
                     else ( (- b - sqrt(delta a b c)) / 2.0 * a,
                            (- b + sqrt(delta a b c)) / 2.0 * a)

Main> primeiro_grau 3 4
-1.333333
Main> segundo_grau 2 (-3) (-4)
(-13.97056,19.97056)
```

---

6) Construa uma função que calcule o valor do mínimo múltiplo comum de três números inteiros.

```
main> mmc 2 3 4
12
```

**RESPOSTA:**

```
mdc::Int->Int->Int
mdc a b | a < b = mdc b a
        | b == 0 = a
        | otherwise = mdc b (mod a b)

mmc::Int->Int->Int
mmc x y = (x * y) `div` (mdc x y)

mmc3n::Int->Int->Int->Int
mmc3n x y z = mmc x (mmc y z)

Main> mmc 12 15
60
Main> mmc3n 12 15 16
240
```

---

7) Construa uma função que calcule o valor do máximo divisor comum entre três números inteiros.

```
main> mdc 2 3 4
1
```

**RESPOSTA:**

```
Usando o Algoritmo de Euclides, podemos escrever:

mdc::Int->Int->Int
mdc a b | a < b = mdc b a
        | b == 0 = a
        | otherwise = mdc b (mod a b)

mdc3n::Int->Int->Int->Int
mdc3n a b c = mdc a (mdc b c)


Main> mdc 2160 888
24
Main> mdc3n 232 2160 888
8
```

---

8) A sequência de Fibonacci é dada pela seguinte série: 0 1 1 2 3 5 8 13 ...
Construa uma função para retornar o n-ésimo termo da sequência.

```
main> fibonacci 6
8
```

**RESPOSTA:**

```
fib::Int->Int
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)

Main> fib 6
8
```

9) Faça uma função que, dado um ano, verifica se o mesmo é bissexto.

**RESPOSTA:**

```
bissexto:: Int-> Bool
bissexto x | (mod x 400 == 0) = True
           | (mod x 4 == 0) && (mod x 100 /= 0) = True
           | otherwise = False

Main> bissexto 2002
False
Main> bissexto 2004
True
```

10) Defina uma função que recebe três números inteiros representando, respectivamente, um dia, um mês e um ano e verifica se os números formam uma data válida.

**RESPOSTA:**

```
type Data = (Int,Int,Int)

valida::Data->Bool
valida (d,m,a)
 | d >= 1 && d <= 31 &&
          (m == 1 || m == 3 || m == 5 || m == 7 || m == 8 ||
          m == 10 || m == 12) = True
 | d >= 1 && d <= 30 &&
          (m == 4 || m == 6 || m == 9 || m == 11) = True
 | d >= 1 && d <= 28 && m == 2 && not (bissexto a) = True
 | d >= 1 && d <= 29 && m == 2 && (bissexto a) = True
 | otherwise = False

Main> valida (1,2,2009)
True
Main> valida (29,2,2002)
False
```