Principios de Lenguajes de Programación

Trabajo Práctico Nº 5: Introducción al lenguaje funcional Haskell

1. Verificar como evalúa `Hugs` las siguientes expresiones:

   (a) $(2 + 3)$
   (b) $\$\$ - 1$ [1]
   (c) $(+)\ 5\ 8$
   (d) $5 - 4 - 3$
   (e) $7\ /\ 3$
   (f) 7 `` `div` `` 3
   (g) 4 `` `div` `` $$
   (h) 7 `` `mod` `` 3
   (i) True || False
   (j) even 9
   (k) succ 6
   (l) succ (pred 6)
   (m) gcd 21 27

2. Dar una definición de la función losCuatrosIguales con el siguiente tipo:

   ```
   Int -> Int -> Int -> Int -> Bool
   ```

   la cual da como resultado `True` si sus cuatros argumentos son iguales.

3. ¿Puede dar una definición de `losCuatroIguales` usando la función `allEqual`?

4. Escriba definiciones de funciones para:

   ```
   cuantosIguales:: Int -> Int -> Int -> Int
   cuantosIgualesDeDos :: Int -> Int -> Int
   ```

   las cuales cuentan si sus argumentos son iguales.

5. Definir las siguientes funciones:

   (a) Que devuelva el enésimo elemento de la serie de Fibonacci.
   (b) El factorial de un número n
   (c) El valor de la siguiente sucesión:

   $$F(x, n) = \frac{\sum_{i=1}^{i=n} i^n}{x!}$$

6. Dar una definición para la función:

   ```
   nAnd :: Bool -> Bool -> Bool
   ```

   que devuelve `True` excepto cuando sus dos argumentos son ambos `True`.

[1] `$$` representa el resultado del último cálculo. En `ghci` puede usarse la expresion `it`.

7. ¿Cómo se podría simplificar esta definición de forma que quede una única cláusula?:

```
funny x y z
   | x  > z      = True
   | x >= y      = False
   | otherwise   = True
```

8. Dar una definición de la función:

   `allDiferent :: Int -> Int -> Int -> Bool`

   se podría utilizar la función "`/=`" con la propiedad que `m /= n` es `True` si `m` y `n` no son iguales. Probar la función `allDiferent` con diferentes valores.

9. ¿Qué encuentra incorrecto en el siguiente código?:

   ```
   AllDiferent n m p = (( n /= m ) && ( m /= p))
   ```

   Probar esta definición con los valores probados en el punto 8)

10. Dar la definición de la función:

    `cuartaPotencia :: Int -> Int`

    Dar otra definición que use `alCuadrado`.

11. Realizar trazas para las siguientes llamadas

    ```
    (max (2+3) (-7)) + (1-3)
    cuantosIgualesDeDos 3 3
    cuantosIguales 1 4 3
    allFourEqual 5 6 4 5
    ```

12. Definir una función tal que dado un `Char` que contenga un dígito, devuelva el valor numérico de dicho dígito.

13. Defina una función

    `digitoRomano :: Char -> String`

    la cual convierte un dígito a su representación en números romanos.

14. Defina una función

    `entreLíneas  :: String -> String -> String -> String`

    la cual tome tres `Strings` y retorne un `String` que cuando se imprima muestre los tres `Strings` en líneas separadas.

15. Defina una función

    `duplicar  :: String -> Int -> String`

la cual tome un `String` y un numero natural `n`. El resultado son `n` copias de un `String` concatenado. (Si `n=0` debe devolver un `String` vacío).

16. Defina una función

    ```
    hacerEspacios :: Int -> String
    ```

    tal que `hacerEspacios n` devuelva un `String` de `n` espacios.

17. Dar una definición para la función

    ```
    factorialTable ::Int -> Int -> String
    ```

    de forma que `factorialTable m n` tabule los valores de los factoriales desde `m` hasta `n` inclusive. Validar los datos de entrada.

18. Definir una función

    ```
    justificarCentro :: Int -> String -> String
    ```

    de forma que `justificarCentro n st` nos devuelva un `String` de longitud `n` en el cual se le han agregado espacios en ambos extremos de `st` de modo que quede centrado. Validar todos los casos.

    Dar una solución utilizando la cláusula `where` y otra sin ella.

19. Definir una función `minMax` la cual retorne el mínimo y el máximo de una tupla.

20. Dar una definición de la función

    ```
    maxOcurr  :: int -> Int -> (Int,Int)
    ```

    el cual retorna el máximo de dos números, junto con el número de veces que aparece. Usando esta u otra función defina

    ```
    maxOcurr  :: int -> Int -> Int -> (Int,Int)
    ```

    que haga lo mismo pero con tres argumentos.

21. Dar una definición para la siguiente función

    ```
    ordenTriple :: (Int,Int,Int) -> (Int, Int,Int)
    ```

    la cual ordena los tres elementos en orden ascendente.

22. Utilizando las definiciones de tipos de Haskell (`type`) definir estructuras de datos convenientes para:

    (a) Un directorio telefónico
    (b) Una base de datos de libros (teniendo en cuenta autores y editoriales)

23. Definir en Haskell una función tal que, dados tres pares de enteros, debe devolver en un cuarto par el menor y el mayor elemento de los pares dados. Por ejemplo para la entrada (2,4) (3,5) (4,1) la salida debe ser (1,5).

24. Definir en Haskell una función tal que, dado un número entero, genere la siguiente salida repitiendo la cantidad de dígitos con una línea para cada dígito. Ejemplo : para el número 1534 , la salida debe ser como:

```
1
55555
333
4444
```

25. Definir en Haskell la función alinear tal que, a partir de tres palabras y un total de caracteres para una línea, debe dar como salida las palabras de manera que una quede justificada a la izquierda, otra centrada y la tercera justificada a la derecha dentro de la misma línea. Por ejemplo:

```
> alinear "hola" "que" "tal" 60
"hola                          que                          tal"
```

26. Definir en Haskell una función tal que, dado un intervalo de años representado por `n m` pueda mostrar en una tabla los años y la leyenda "es bisiesto" o "no bisiesto". Un año es bisiesto si es múltiplo de 4 (por ej. 1984), pero si el año es múltiplo de 100 sólo son bisiestos cuando a su vez son múltiplos de 400. Por ej 1800 no es bisiesto, mientras que 2000 si. Ej: para el intervalo 2000 2004 la tabla sería:

```
2000  bisiesto
2001  no bisiesto
2002  no bisiesto
2003  no bisiesto
2004  bisiesto
```

27. Definir en Haskell una función que dada una tripla `(a,b,c)`, genere una cadena con los números comprendidos entre `a` y `b` que cumplan con la propiedad de ser múltiplos de `c`. Ej: para la tripla `(3, 20, 6)` debe generar "`6 12 18`"

28. Definir en Haskell una función tal que, dado un intervalo expresado como (n,m) genere una tabla con los números del intervalo que sean cuadrados perfectos y la posición que ocupan dentro del intervalo. Por ejemplo:

```
> intervalo (2,18)
4  3
9  8
16 15
```