UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación avanzada II

Lab 1. Un primer contacto con Scala

1. Escribe una función recursiva de cola `primeFactors(n: Int): List[Int]` que devuelva una lista con los factores primos de un entero positivo dado n. Ejemplos:

```
println(primeFactors(60))   // Output: List(2, 2, 3, 5)
println(primeFactors(97))   // Output: List(97)
println(primeFactors(84))   // Output: List(2, 2, 3, 7)
```

2. Escribe una función recursiva de cola `binarySearch(arr: Array[Int], elt: Int): Option[Int]` que devuelva el índice de elt (`Some(i)`) en un array ordenado utilizando el algoritmo de búsqueda binaria, o None en caso de que el elemento no se esté. Ejemplos:

```
val arr = Array(1, 3, 5, 7, 9, 11)
println(binarySearch(arr, 5))  // Output: Some(2)
println(binarySearch(arr, 10)) // Output: None
```

3. Define una función recursiva genérica `unzip` que tome una lista de tuplas con dos componentes y que devuelva una tupla con dos listas: una con las primeras componentes y otra con las segundas. Por ejemplo,

```
unzip(List((10, 'a'), (20, 'b'), (10, 'c'))
== (List(10, 20, 30), List('a', 'b', 'c'))
```

4. Define una función recursiva genérica `zip` que tome dos listas y devuelva una lista de tuplas, donde las primeras componentes se tomen de la primera lista y las segundas componentes de la segunda lista. Por ejemplo:

```
zip(List(10, 20, 30), List('a', 'b', 'c'))
== List((10, 'a'), (20, 'b'), (10, 'c'))
zip(List(10, 20, 30), List('a', 'b'))
== List((10,'a'), (20,'b'))
```

5. Implementa una operación filter(l, f) que tome una lista l de elementos de tipo A y una función f: A => Boolean y que devuelva una lista con los elementos e de l que satisfacen f(e). Por ejemplo:

```
println(filter(List(1,2,3,4,5), _ % 2 == 0)) // Output: List(2,4)
```

6. Implementa una operación map(l, f) que tome como argumentos una lista l de elementos de tipo A y una función f: A => B y que devuelva una lista de elementos de tipo B con los elementos resultantes de aplicar f a cada uno de los elementos de l.

```
println(map(List(1,2,3,4,5), _ * 2)) // Output: List(2,4,6,8,10)
```

7. Implementa una operación groupBy(l, f) que tome como argumentos una lista l de elementos de tipo A y una función f: A => B y que devuelva un objeto de tipo Map[B, List[A]] que asocie una lista con los elementos e de l con el mismo f(e).

```
println(groupBy(List(1,2,3,4,5), _ % 2 == 0))
      // Output: Map(false -> List(5, 3, 1), true -> List(4, 2))
```

8. Implementa una operación reduce(l, f) que toma como argumentos una lista l de elementos de tipo A y una función f de tipo (A, A) => A y que devuelva el resultado de combinar todos los elementos de l utilizando la función f. Por ejemplo:

```
println(reduce(List(1,2,3,4,5), _ + _)) // Output: 15
```

9. Implementa una función recursiva para generar todos los subconjuntos de un conjunto determinado. Conviértela en recursiva de cola.

```
println(subsets(Set())) // Output: Set(Set())
println(subsets(Set(1))) // Output: Set(Set(), Set(1))
println(subsets(Set(1,2))) // Output: Set(Set(),Set(1),Set(2),Set(1,2))
println(subsets(Set(1, 2, 3)))
// Output: Set(Set(),Set(1),Set(2),Set(1,2),Set(3),Set(1,3),Set(2,3),Set(1,2,3))
```

10. Escribe una función recursiva de cola `generateParentheses(n: Int): List[String]` que genere todas las combinaciones válidas de n pares de paréntesis. Ejemplos:

```
println(generateParentheses(3))
// Output: Lista("((()))", "(()())", "(())()", "()(())", "()()()")
```

Consejos:

- Utiliza un acumulador para almacenar secuencias válidas.
- Haz un seguimiento del número de paréntesis de apertura (open) y cierre (closed) utilizados.
- Caso base: Cuando open == closed == n, agrega la secuencia al resultado.