UNIVERSIDAD DE MÁLAGA | uma.es

Dpt. Languages and Computer Sciences

Advanced Programming II

Lab 1. A first contact with Scala

1. Write a tail-recursive function `primeFactors(n: Int): List[Int]` that returns a list of the prime factors of a given positive integer n. Examples:

```
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. Write a tail-recursive function `binarySearch(arr: Array[Int], elt: Int): Option[Int]` that returns either the index of the item elt (`Some(i)`) in a sorted array using the binary search algorithm, or `None` in case the element is missing. Examples:

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

3. Define a generic recursive function **`unzip`** that takes a list of tuples with two components and return a tuple with two lists: one with the first components and another with the second ones. For example:

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

4. Define a generic recursive function **`zip`** that takes two lists and returns a list of tuples, where the first components are taken from the first list and the second components from the second list. For example:

```
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. Implement an operation `filter(l, f)` that takes a list l of elements of type A and a function `f: A => Boolean` and returns a list of the elements e from l that satisfy f(e). For example:

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

6. Implement an operation `map(l, f)` that takes as arguments a list l of elements of type A and a function f: A => B and returns a list of elements of type B with the elements resulting from applying f to each of the elements of l. For example:

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

7. Implement an operation `groupBy(l, f)` that takes as arguments a list l of elements of type A and a function `f: A => B` and returns an object of type Map[B, List[A]] that associates a list with the elements e of l with the same `f(e)`. For example:

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

8. Implement an operation `reduce(l, f)` that takes as arguments a list l of elements of type A and a function f of type `(A, A) => A` and returns the result of combining all the elements of l using the function f. For example:

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

9. Implements a recursive function to generate all subsets of a given set. Make it tail-recursive.

```
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. Write a tail-recursive function `generateParentheses(n: Int): List[String]` that generates all valid combinations of n pairs of parentheses. Examples:

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

Hints:

- Use an accumulator to store valid sequences.
- Keep track of the number of open and closed brackets already used.
- Base case: when open == closed == n, you should adds the sequence to the accumulator.