Programmazione Funzionale e Parallela (A.A. 2015-2016)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

D

Esonero dell'11/12/2015 – Durata 1h 30'

Inserire nome, cognome e matricola nel file `studente.txt`. E' possibile usare Eclipse oppure un qualsiasi editor di testo usando `scalac`/`scala` da riga di comando.

Esercizio 1

Si vuole estendere il linguaggio Scala con un nuovo costrutto `alterna` che, dato un intero e due blocchi `Unit`, alterna per n volte la valutazione del primo blocco con la valutazione del secondo blocco.

Scrivere la soluzione in un file `D1.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `D1Main.scala`:

```
import D1._

object D1Main extends App {
    alterna (3) { print("uno") } { print("due ") }
    println(" [corretto: \"unodue unodue unodue \"]")

    alterna (0) { print("uno") } { print("due ") }
    println(" [corretto: \"\"]")

    val alternaTre:(=>Unit)=>(=>Unit)=>Unit = alterna(3)

    alternaTre { print("A") } { print("B") }
    println(" [corretto: \"ABABAB\"]")
}
```

La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare alcuna variabile `var`.

Esercizio 2

Si vuole scrivere un metodo che fonde due liste di stringhe in un'unica lista in cui ogni elemento è ottenuto concatenando i rispettivi elementi nelle due liste. Se una delle due liste è più corta dell'altra, la lista risultante verrà completata con gli elementi rimanenti della lista più lunga. Scrivere la soluzione in un file `D2.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `D2Main.scala`:

```
object D2Main extends App {

    val l1 = List("uno", "due", "tre")
    val l2 = List("1", "2", "3", "4")
    val l3 = List("1", "2")

    val l4:List[String] = D2.combineLists(l1, l2)
    println(l4+" [corretto: List(uno1, due2, tre3, 4)]")

    val l5 = D2.combineLists(l1, l3)
    println(l5+" [corretto: List(uno1, due2, tre)]")

    val l6 = D2.combineLists(l1, Nil)
    println(l6+" [corretto: List(uno, due, tre)]")
}
```

La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare alcuna variabile `var`.

Esercizio 3

Si richiede di scrivere un metodo `makeTree` che genera un albero binario completo con k livelli. Ad ogni nodo dell'albero generato deve essere associata un'etichetta intera: la radice ha etichetta 1 e un nodo di etichetta i ha come figlio sinistro il nodo di etichetta 2i e come figlio destro il nodo di etichetta 2i+1.

Il metodo prende come parametri il numero k di livelli dell'albero (0=albero vuoto, 1=albero con un solo nodo, ecc.) e restituisce un albero completo con k livelli dove il nodo di etichetta i contiene il valore i.

Scrivere la soluzione estendendo il file `D3.scala`:

```
sealed abstract class Tree()
case class E() extends Tree() {
    override def toString = "-"
}
case class T(l:Tree, x:Int, r:Tree) extends Tree() {
    override def toString = "["+l+","+x+","+r+"]"
}
```

in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `D3Main.scala`:

```
object D3Main extends App {

    val t1:Tree = D3.makeTree(2)
    println(t1+" [corretto: [[-,2,-],1,[-,3,-]]]")

    val t2 = D3.makeTree(1)
    println(t2+" [corretto: [-,1,-]]")

    val t3 = D3.makeTree(3)
    println(t3+
      " [corretto: [[[-,4,-],2,[-,5,-]],1,[[-,6,-],3,[-,7,-]]]]")

    val t4 = D3.makeTree(0)
    println(t4+" [corretto: -]")
}
```

La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare alcuna variabile `var`.