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

B

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 scrivere una classe che rappresenta punti nel piano con le operazioni di somma vettoriale di punti e distanza tra punti. Per il calcolo della distanza, usare la formula $dist\big((x_1, y_1), (x_2, y_2)\big) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Si noti che la radice quadrata può essere calcolata tramite il metodo predefinito: `Math.sqrt(x:Double):Double`. Inoltre, si ha $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$.

Scrivere la soluzione in un file `B1.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `B1Main.scala`:

```
import Punto._

object B1Main extends App {
    val p1:Punto = Punto(1,3)
    println(p1 + " [corretto: (1.0,3.0)]")

    val p2:Punto = 10
    println(p2 + " [corretto: (10.0,10.0)]")

    val p3:Punto = p1 + p2
    println(p3 + " [corretto: (11.0,13.0)]")

    val p4:Double = p1 <-> p3
    println(p4 + " [corretto: 14.142135623730951]")
}
```

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

Esercizio 2

Si vuole estendere la classe `String` con un metodo Scala che ripete ogni carattere di una stringa un certo numero positivo di volte.

Scrivere la soluzione in un file `B2.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `B2Main.scala`:

```
import B2._

object B2Main extends App {
    val s1:String = "Leia" --> 2
    println(s1+""" [corretto="LLeeiiaa"]""")

    val s2:String = "Yoda" --> 3
    println(s2+""" [corretto="YYYooodddaaa"]""")

    val s3:String = "Anakin" --> 1
    println(s3+""" [corretto="Anakin"]""")
}
```

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

Esercizio 3

Si vuole scrivere un metodo `min` che trova il minimo in un albero binario di ricerca. Si ricordi che in un albero binario di ricerca, dato un qualsiasi nodo v che contiene una chiave k, ogni chiave nel sottoalbero sinistro di v è minore o uguale a k e ogni chiave nel sottoalbero destro di v è maggiore o uguale a k. Per cercare il minimo, basta scendere nell'albero andando sempre a sinistra finché possibile. Il metodo `min` deve restituire una coppia (c, d), dove c è il livello su cui si trova il minimo (assumendo che il livello della radice sia 1) e d è il valore del minimo. Su un albero vuoto, `min` deve restituire (0, 0).

Scrivere la soluzione in un file `B3.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `B3Main.scala`:

```
object B3Main extends App {
    val t:Tree = T(T(E(),5,T(E(),7,E())),10,T(E(),15,T(E(),17,E())))

    val p1:(Int,Int) = t.min                /*    10     */
    println(p1+" [corretto: (2,5)]")        /*   /  \    */
                                            /*  5   15   */
    val p2:(Int,Int) = E().min              /*   \    \  */
    println(p2+" [corretto: (0,0)]")        /*    7   17 */

    val p3:(Int,Int) = T(E(),5,E()).min
    println(p3+" [corretto: (1,5)]")
}
```

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