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

C

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 `ifElse` che, data una condizione booleana e due blocchi `Unit`, valuta il primo se la condizione è vera e valuta il secondo se la condizione è falsa.

Scrivere la soluzione in un file `C1.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `C1Main.scala`:

```
import C1._

object C1Main extends App {
    val x = 5
    ifElse (x<10) { print("if") } { print("else") }
    println(" [corretto: if]")

    ifElse (x>10) { print("if") } { print("else") }
    println(" [corretto: else]")
}
```

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

Esercizio 2

Si vuole scrivere una funzione `makeList` che, data una funzione f che mappa interi su elementi generici e un intero n, restituisce una lista contenente n elementi dove l'elemento di indice i è dato da f(i).

Scrivere la soluzione in un file `C2.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `C2Main.scala`:

```
object C2Main extends App {
    val l1:List[String] =
        C2.makeList(i=>if (i<1) "TEST" else "test")(3)
    println(l1+" [corretto: List(TEST, test, test)]")

    val altern01: Int=>List[Int] =
        C2.makeList(i=>if (i%2==0) 0 else 1)

    val l2 = altern01(5)
    println(l2+" [corretto: List(0, 1, 0, 1, 0)]")
}
```

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

Esercizio 3

Si vuole scrivere una funzione che verifica l'uguaglianza di alberi binari, definendo il metodo `equals` su alberi.

Scrivere la soluzione estendendo il file `C3.scala`:

```
sealed abstract class Tree
case class E() extends Tree()
case class T(l:Tree, x:Int, r:Tree) extends Tree()
```

in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `C3Main.scala`:

```
object C3Main extends App {
    val e  = E()
    val t1 = T(T(T(e,9,e),6,e),1,T(e,5,e))
    val t2 = T(T(T(e,9,e),6,e),1,T(e,5,e))
    val t3 = T(e,10,e)

    val b1:Boolean = t1 == t2
    println(b1+" [corretto: true]")

    val b2:Boolean = e == E()
    println(b2+" [corretto: true]")

    val b3:Boolean = t1 == t3
    println(b3+" [corretto: false]")

    val b4:Boolean = t3 == T(e,10,e)
    println(b4+" [corretto: true]")

    val b5:Boolean = e == t3
    println(b5+" [corretto: false]")
}
```

La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare alcuna variabile `var`. Si noti che il metodo == in Scala richiama il metodo `equals`.