Programmazione Funzionale e Parallela (A.A. 2016-2017)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

A

Esonero del 14/12/2016 – 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 la classe `Set[Int]` con le operazioni di unione (`+`) e differenza (`-`). Si ricordi che, se `a` e `b` sono due oggetti `Set`, `a.union(b)` è la loro unione e `a.diff(b)` è la loro differenza.

Scrivere la soluzione in un file `A1.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `A1Main.scala`:

```
import MyRichSet._

object A1Main extends App {

    val s1 = Set(1,2,3,4)
    val s2 = Set(4,3,5,7)

    val s3:Set[Int] = s1 + s2 // unione insiemi
    println(s3 + " [corretto: 1,2,3,4,5,7 (in qualsiasi ordine)]")

    val s4:Set[Int] = s1 - s2 // differenza insiemi
    println(s4 + " [corretto: 1,2  (in qualsiasi ordine)]")
}
```

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

Esercizio 2

Si richiede di scrivere codice Scala per effettuare analisi di dati.

Scrivere la soluzione in un file `A2.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `A2Main.scala`:

```
case class Studente(val nome:String, val età:Int, val esami:List[String])
object A2Main extends App {
    val q = List(
              Studente("Marco", 24, List("PFP","SC")),
              Studente("Laura", 19, List("SC", "FI1", "PFP", "DB")),
              Studente("Stefano", 23, List("SC", "FI1")),
              Studente("Marco", 25, List("SC", "FI1", "FI2")),
              Studente("Paola", 21, List("SC", "PFP")),
              Studente("Lucia", 18, List("SC", "PFP", "OOP"))
            )

    // query1 estrae la lista di tutti gli studenti che hanno
    // età inferiore alla media e hanno sostenuto almeno tre esami
    val query1:List[Studente] = A2.query1(q)
    println(query1.map(_.nome))
    println("--> [corretto: Laura e Lucia]")

    // query2 estrae la lista di tutti gli esami che sono stati
    // sostenuti da almeno due studenti
    val query2:List[String] = A2.query2(q)
    println(query2)
    println("--> [corretto: SC, FI1 e PFP]")
}
```

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

Esercizio 3

Si richiede di scrivere una gerarchia di classi Scala per rappresentare expressioni booleane con due incognite.

Scrivere la soluzione in un file `A3.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `A3Main.scala`:

```
object A3Main extends App {

    // parte I: definizione della gerarchia e conversione a stringa di exp
    val e:Exp = And(Or(X(), Y()), Not(Y()))
    println(e + " [corretto: and(or(x,y),not(y))")

    // parte II: valutazione di espressioni booleane
    for ( x <- List(true, false); y <- List(true, false) ) {
        val b:Boolean = e(x,y)
        println("e("+x+","+y+")="+b+" [corretto: "+((x||y) && !y)+"]")
    }

    // parte III: relazione se e solo se (<=>)
    val e1:Exp = And(Or(X(), Y()), Not(Y()))
    val e2:Exp = Not(Or(And(Not(X()), Not(Y())),Y()))
    val e3:Exp = And(Or(X(), Not(X())),Or(Y(), Not(Y())))
    println("e1 <=> e2: "+(e1 <=> e2)+" [corretto: true]")
    println("e1 <=> e3: "+(e1 <=> e3)+" [corretto: false]")
    println("True() <=> e3: "+(True() <=> e3)+" [corretto: true]")
    println("True() <=> False(): "+(True() <=> False())+
            " [corretto: false]")
}
```

La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare alcuna variabile `var`. **Suggerimento:** sviluppare in ordine le tre parti richieste indicate nel testo, commentando le parti non ancora svolte in modo da svolgere l'esercizio in modo incrementale compilando e testando mano mano il codice.