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

B

Esonero del 16/12/2016 – 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 arricchire la classe `Exp` per espressioni booleane contenuta nel file `B1.scala` con un metodo che consenta di **semplificare** un'espressione booleana tenendo conto delle usuali regole che coinvolgono $true$, $false$, $\wedge$, $\vee$ e $\neg$ (es. l'$\wedge$ con un operando $false$ è $false$, il $\neg$ di $false$ è $true$, ecc.). Ad esempio, la formula $\neg(false \wedge \neg y) \wedge \neg(x \vee false)$ si può semplificare in $\neg x$.

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

```
object B1Main extends App {
    def test(e:Exp, c:Exp) {
        val s:Exp = e.simplify
        println(s + " ["+ (if (c==s) "ok"
                           else "errore, dovrebbe essere: "+c) +"]")
    }
    test( And(Not(False()),X()), X() )
    test( And(X(),True()), X() )
    test( And(X(),False()), False() )
    test( And(False(),X()), False() )
    test( Or(False(),X()), X() )
    test( Or(X(),False()), X() )
    test( Or(X(),True()), True() )
    test( Or(True(),X()), True() )
    test( And(Not(And(False(),Not(Y()))), Not(Or(X(),False()))), Not(X()))
}
```

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

Esercizio 2

Si richiede di aggiungere a Scala un costrutto `mioFor` che simula il funzionamento di un costrutto for imperativo classico come nell'esemio sotto.

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 {
    var i = 1
    mioFor(i <= 10, i += 1) { print(i+" ") }
    println("[corretto: 1 2 3 4 5 6 7 8 9 10]")
}
```

Esercizio 3

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

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

```
case class Studente(val nome:String, val età:Int, val esami:List[String])

object B3Main 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("Marina", 25, List("SC", "FI1", "FI2")),
              Studente("Paola", 21, List("SC", "PFP")),
              Studente("Lucia", 18, List("SC", "PFP", "OOP"))
            )

    // query1 produce una lista di coppie
    // (esame, lista nomi studenti con almeno 22 anni che l'hanno superato)
    val query1:List[(String, List[String])] = B3.query1(q)
    println(query1)
    println("--> [corretto: List((SC,List(Marco, Stefano, Marina)), "+
                  "(FI1,List(Stefano, Marina)), (PFP,List(Marco)), "+
                  "(FI2,List(Marina))) (in qualsiasi ordine)]")

    // query2 produce la lista di tutte le coppie di esami per cui esistono
    // almeno due studenti che li ha superati entrambi
    val query2:List[(String,String)] = B3.query2(q)
    println(query2)
    println("--> [corretto: List((PFP,SC), (FI1,SC))]")
}
```

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