Programmazione Funzionale e Parallela

Corso di Laurea in Ingegneria Informatica e Automatica - A.A. 2019-2020

Esercitazione del 16 marzo 2020

Istruzioni per l'esercitazione:

- Aprite il form di consegna in un browser e loggatevi con le vostre credenziali `uniroma1`.
- Scaricate e decomprimete sulla scrivania il codice dell'esercitazione. Vi sarà una sotto-directory separata per ciascun esercizio di programmazione. Non modificate in alcun modo i programmi di test `E*Main.scala`.
- Rinominare la directory chiamandola `cognome.nome`. Sulle postazioni del laboratorio sarà `/home/studente/Desktop/cognome.nome/`.
- È possibile consultare appunti/libri e il materiale didattico online.
- Rispondete alle domande online sul modulo di consegna.
- **Finiti gli esercizi**, e non oltre le 23:00:
  - **zippate la directory di lavoro** in `cognome.nome.zip` (`zip -r cognome.nome.zip cognome.nome/`).
- **Per consegnare**:
  - inserite nel form di consegna come autovalutazione il punteggio di ciascuno dei test forniti (inserite zero se l'esercizio non è stato svolto, non compila, o dà errore di esecuzione).
  - fate **upload** del file `cognome.nome.zip`.
- È possibile consultare la documentazione delle API di Scala, in particolare quelle sulle liste, e la dispensa Scala.
- **Se avete domande** accedete a Google Meet all'indirizzo meet.google.com/rey-xdve-rug durante orario 14:00-16:00 stabilito per l'esercitazione accedendo con la vostra **mail istituzionale**. Troverete online il docente e il tutor del corso. In alternativa, scrivete via email.

Per maggiori informazioni fate riferimento al regolamento delle esercitazioni.

Esercizio 1 (somma di funzioni)

Scrivere una funzione `sommaFun(f1:Double=>Double, f2:Double=>Double):Double=>Double` che restituisce la funzione somma di `f1` ed `f2`. Ad esempio: `sommaFun(x=>x, x=>x+1)(2) == 5` (ottenuto come: `2+(2+1)`), `sommaFun(x=>2*x, x=>x+2)(3) == 11` (ottenuto come: `(2*3)+(3+2)`)

Per compilare da riga di comando usare: `scalac E1Main.scala E1.scala`. Si noti che sulla riga di comando ci sono entrambi i file che compongono il programma. Noterete la presenza di vari file `.class` generati dalla compilazione.

Per eseguire il programma da riga di comando usare: `scala E1Main`. Si noti che, come in Java, al comando `scala` viene passato il nome della classe.

Esercizio 2 (corrispondenza di liste)

Scrivere una funzione `corrisp[A,B](a:List[A], b:List[B], f:A=>B):Boolean` che restituisce `true` se e solo se per ogni indice `i` comune a entrambe le liste vale `b(i)=f(a(i))`. Se una lista è più lunga dell'altra, gli elementi in eccedenza devono essere ignorati.

Scrivere la soluzione nel file `E2.scala` e usare il programma di prova `E2Main.scala`.

Esercizio 3 (prefissi di liste)

Scrivere una funzione `maxPrefisso(l:List[Int], x:Int):Int` Scala che restituisce il più grande numero `n` tale che la somma dei primi `n` numeri di `l` è minore o uguale a `x`. Ad esempio, `maxPrefisso(List(1,1,1,1),3) == 3`, `maxPrefisso(List(5,2,4,7),8)==2` e `maxPrefisso(List(5,2,4,7),4)==0`.

Scrivere la soluzione nel file `E3.scala` e usare il programma di prova `E3Main.scala`.

Esercizio 4 (sequenze bitoniche)

Una *sequenza bitonica* è formata da una sequenza non vuota strettamente crescente seguita da una sequenza non vuota strettamente decrescente, ad esempio: `List(1,2,5,6,9,4,3,2,0)` è bitonica, mentre `List(1,2,3,2,3,2,1)`, `List(1,2,3)` e `List()` non lo sono.

Scrivere una funzione `checkBitonic(l:List[Int]):(List[Int],List[Int])` che, data una lista `l` bitonica, restituisce `(inc,dec)` tale che `inc` è il prefisso crescente di `l` che include l'elemento massimo e `dec` è il suffisso strettamente decrescente che segue (si ha che `inc ::: dec == l`). Se invece `l` non è bitonica, la funzione restituisce `(Nil,Nil)`.

Scrivere la soluzione nel file `E4.scala` e usare il programma di prova `E4Main.scala`.