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

A

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 Scala per la gestione dei numeri complessi. Si ricordi che:

- un numero complesso è una coppia a+ib dove a è la parte reale e b è la parte immaginaria;
- la somma di due numeri complessi è (a+ib)+(c+id) = (a+c)+i(b+d);
- il prodotto di due numeri complessi è (a+ib)(c+id) = (ac-bd)+i(bc+ad).

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 Complex._
object A1Main extends App {
    val c1:Complex = Complex(1,3)
    println(c1 + " [corretto: 1.0+i3.0]")

    val c2:Complex = 10
    println(c2 + " [corretto: 10.0+i0.0]")

    val c3:Complex = c1 + c2
    println(c3 + " [corretto: 11.0+i3.0]")

    val c4:Complex = c1 * c3
    println(c4 + " [corretto: 2.0+i36.0]")
}
```

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

Esercizio 2

Si richiede di scrivere una funzione Scala che, dati come parametri una funzione c che mappa interi su caratteri e un intero n, genera una stringa di lunghezza n in cui il carattere di indice i è pari a c(i).

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

```
object A2Main extends App {
    val s1:String = A2.makeStr(i=>if (i%2==0) '/' else '\\')(10)
    println(s1+""" [corretto="/\/\/\/\/\"]""")

    val makeLineStr:Int=>String = A2.makeStr(i=>'-')

    val s2 = makeLineStr(5)
    println(s2+""" [corretto="-----"]""")

    val s3 = A2.makeStr(i=>"tes".charAt(i%3))(13)
    println(s3+""" [corretto="testestestest"]""")
}
```

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

Esercizio 3

Si vuole scrivere un metodo ricorsivo `find` che effettua la ricerca di una chiave 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 un elemento x, basta verificare se coincide con quello y contenuto nella radice. In caso contrario, si andrà ricorsivamente a sinistra se x<y e a destra se x>y. Il metodo `find` deve restituire una coppia (c, b), dove c è il numero di chiamate a `find` effettuate durante la ricerca di un elemento e b vale `true` se e solo se l'elemento è stato trovato.

Estendere il file `A3.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 `A3Main.scala`:

```
object A3Main extends App {
    val t:Tree = T(T(E(),5,T(E(),7,E())),10,T(E(),15,E()))

    val b1:(Int,Boolean) = t find 7         /*     10     */
    println(b1+" [corretto: (3,true)]")     /*    /  \    */
                                            /*   5   15   */
    val b2 = t find 6                       /*    \       */
    println(b2+" [corretto: (4,false)]")    /*     7      */

    val b3 = t find 15
    println(b3+" [corretto: (2,true)]")

    val b4 = t find 13
    println(b4+" [corretto: (3,false)]")

    val b5 = t find 10
    println(b5+" [corretto: (1,true)]")
}
```

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