# Programmazione Funzionale e Parallela (A.A. 2015-2016)

Corso di Laurea in Ingegneria Informatica e Automatica Sapienza Università di Roma

### Esame del 10/02/2016 – Durata 1h 30' (non esonerati)

Inserire nome, cognome e matricola nel file studente.txt.

### Esercizio 1

Si vuole scrivere un metodo Scala minmax che calcola simultaneamente minimo e massimo di un albero binario di ricerca e il numero di chiamate ricorsive effettuate. Si ricordi che in un albero binario di ricerca la chiave di ogni nodo è maggiore o uguale alle chiavi nel suo sottoalbero sinistro e minore o uguale alle chiavi nel suo sottoalbero destro. La soluzione deve usare il minor numero possibile di chiamate ricorsive.

Completare il seguente file C1.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 ClMain.scala:

La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare alcuna variabile var. *Suggerimento*: può essere utile sfruttare le costanti Java Integer.MIN\_VALUE e Integer.MAX\_VALUE, che denotano il minimo e il massimo del dominio degli Int.

#### Esercizio 2

Si vuole scrivere un metodo che calcola il numero di spazi di una stringa. Scrivere la soluzione nel file C2.scala in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova C2Main.scala:

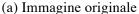
```
object C2Main extends App {
    val p1:Int = C2.countSpaces("hi there!")
    println(p1+" [corretto: 1]")
    val p2:Int = C2.countSpaces(" one two three ")
    println(p2+" [corretto: 4]")
    val p3:Int = C2.countSpaces("zero")
    println(p3+" [corretto: 0]")
}
```

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

## Esercizio 3 (Aggiunta bordo a immagine mediante OpenCL)

Lo scopo dell'esercizio è quello di scrivere un modulo C basato su OpenCL che, data in input un'immagine a 256 toni di grigio di dimensione  $w \times h$ , crei una nuova immagine di dimensioni  $(w + 2f) \times (h + 2f)$ , dove f è lo spessore del bordo, come nell'esempio sotto.







(b) Immagine dopo aggiunta bordo

Si completi nel file addframe/addframe.c la funzione addframe con il seguente prototipo:

#### dove:

- in: puntatore a un buffer di dimensione w\*h\*sizeof(unsigned char) byte che contiene l'immagine di input in formato row-major<sup>1</sup>;
- w: larghezza di in in pixel (numero di colonne della matrice di pixel);
- h: altezza di in in pixel (numero di righe della matrice di pixel);
- frame grey: tono di grigio del bordo;
- frame size: spessore del bordo in pixel;
- out: puntatore a puntatore a buffer di dimensione (w+2\*frame\_size)\* (h+2\*frame\_size)\*sizeof(unsigned char) byte che deve contenere l'immagine di output in formato row-major; il buffer deve essere allocato nella funzione addframe:
- ow: puntatore a int in cui scrivere la larghezza di out in pixel;
- oh: puntatore a int in cui scrivere l'altezza di out in pixel.

Per compilare usare il comando make. Per effettuare dei test usare make test1 e make test2. Verranno prodotte immagini di output colosseo-addframe\*.pgm.

<sup>&</sup>lt;sup>1</sup> Cioè con le righe disposte consecutivamente in memoria.