

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

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

C

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 `C1Main.scala`:

```
object C1Main extends App {
  val t:Tree = T(T(E(),5,T(E(),7,E())),10,T(E(),15,T(E(),17,E())))

  val p1:(Int,Int,Int) = t.minMax           /*      10      */
  println(p1+" [corretto: (5,17,5)]")       /*    /  \     */
                                           /*   5   15    */
  val p2:(Int,Int,Int) = T(t,25,E()).minMax /*    \      \  */
  println(p2+" [corretto: (5,25,4)]")       /*     7     17 */

  val p3:(Int,Int,Int) = T(E(),9,E()).minMax
  println(p3+" [corretto: (9,9,2)]")
}
```

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.



(a) Immagine originale



(b) Immagine dopo aggiunta bordo

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

```
void addframe(unsigned char* in, int w, int h,
              unsigned char frame_grey, int frame_size,
              unsigned char** out, int* ow, int* oh,
              clut_device* dev, double* td);
```

dove:

- `in`: puntatore a un buffer di dimensione $w \times h \times \text{sizeof}(\text{unsigned char})$ byte che contiene l'immagine di input in formato row-major¹;
- `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 $(w+2 \times \text{frame_size}) \times (h+2 \times \text{frame_size}) \times \text{sizeof}(\text{unsigned char})$ 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`.

¹ Cioè con le righe disposte consecutivamente in memoria.