Programmazione Funzionale e Parallela (A.A. 2018-19)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

A

Esame del 22/07/2019 – Durata 1h 30'

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

Esercizio 1 (Scala)

Si vuole scrivere un metodo Scala `select` che, data una lista di automobili e una lista di proprietari, trova per ogni proprietario il modello della sua automobile più nuova. Assumere che ogni proprietario nella lista di input possegga almeno un'auto.

Scrivere la soluzione in un file `A1.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `A1Main.scala`:

```
case class Car(model:String, year:Int, owner:String)

object A1Main extends App {
    val cars = List(Car("Tesla Model X", 2017, "Elon"),
                    Car("Open Zafira", 2008, "Anna"),
                    Car("Audi Quattro", 2013, "Ron"),
                    Car("Rover 220 SDI", 1999, "Anna"))
    val owners = List("Elon", "Anna")
    val res:List[String] = A1.select(cars, owners)
    println(res + " - corretto: List(Tesla Model X, Open Zafira)")
}
```

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

Esercizio 2 (Scala)

Si vuole estendere la classe `Int` con un metodo Scala che calcola simultaneamente il massimo e il minimo di due numeri. Scrivere la soluzione in un file `A2.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `A2Main.scala`:

```
import A2._

object A2Main extends App {
    val r1:(Int,Int) = 8 minMax 5
    println(r1+" [corretto=(5,8)]")
    val r2 = 1 minMax 2
    println(r2+" [corretto=(1,2)]")
}
```

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

Esercizio 3 (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 $w/2 \times h/2$, ottenuta riscalando l'immagine originale del 50%. Il pixel di coordinate (x,y) della matrice di output sarà ottenuto come media aritmetica dei valori di grigio dei quattro pixel della matrice di input di coordinate (2x, 2y), (2x+1, 2y), (2x, 2y+1) e (2x+1, 2y+1). Esempio:

(a) Immagine originale

(b) Immagine dimezzata

Si completi nel file `scale2x/scale2x.c` la funzione `scale2x` con il seguente prototipo:

```
void scale2x(unsigned char* in, int w, int h,
             unsigned char** out, int* ow, int* oh,
             clut_device* dev, double* td)
```

dove:

- `in`: puntatore a un buffer di dimensione `w*h*sizeof(unsigned char)` byte che contiene l'immagine di input in formato row-major[1];
- `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);
- `out`: puntatore a puntatore a buffer di dimensione `w/2*h/2*sizeof(unsigned char)` byte che deve contenere l'immagine di output in formato row-major; **il buffer deve essere allocato nella funzione scale2x**;
- `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 un test usare `make test`. Verrà prodotta l'immagine di output `colosseo-poster.pgm`.

[1] Cioè con le righe disposte consecutivamente in memoria.