Programmazione Funzionale e Parallela (A.A. 2016-2017)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

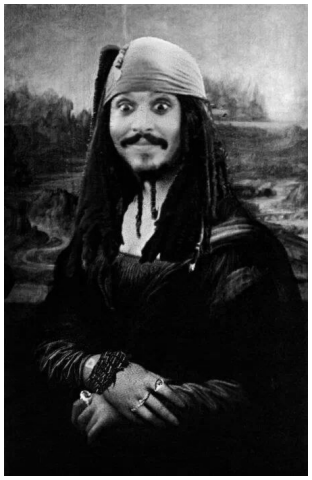
A

Esame del 10/04/2017 – Durata 1h 45' (non esonerati)

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

Esercizio 1 (Filtri grafici 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 $wc \times hc$ ottenuta ritagliando una porzione dell'immagine come nell'esempio sotto:



(a) Immagine di input

(b) Immagine di output

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

```
void crop(unsigned char* in,
          unsigned xc, unsigned yc, unsigned wc, unsigned hc,
          unsigned w,  unsigned h, unsigned char** out,
          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];
- `xc`: coordinata x dell'angolo superiore sinistro della porzione di `in` da ritagliare;
- `yc`: coordinata y dell'angolo superiore sinistro della porzione di `in` da ritagliare;
- `wc`: larghezza di `out` in pixel (numero di colonne della matrice di pixel);
- `hc`: altezza di `out` in pixel (numero di righe della matrice di pixel);
- `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 `wc*hc*sizeof(unsigned char)` byte che deve contenere l'immagine di output in formato row-major; **il buffer deve essere allocato nella funzione `crop`**.

Per compilare usare il comando `make`. Per effettuare un test usare `make test`. Verrà prodotta l'immagine di output `johnny-crop.pgm`.

---

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

Esercizio 2 (Elimina prefisso e suffisso)

Scrivere una funzione `trim` che, data una stringa, resituisce la stessa stringa dove è stato eliminato il prefisso e il suffisso che soddisfa un dato predicato. Ad esempio, `A2.trim("ada**babb**da", c=>c=='a'||c=='d')` restituisce `"babb"`. Si veda il programma di prova sotto per altri esempi.

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 {
    // test 1
    val m1 = A2.trim("  in vino veritas  ", _ == ' ');
    println("\"" + m1 + "\" [corretto: \"in vino veritas\"]")

    // test 2
    val m2 = A2.trim("[This is a test...]", !_.isLetter);
    println("\"" + m2 + "\" [corretto: \"This is a test\"]")
}
```

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

Esercizio 3 (Verifica se due liste sono disgiunte)

Scrivere una funzione generica `test` che se due liste sono insiemisticamente disgiunte.

Scrivere la soluzione in un file `A3.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `A3Main.scala`:

```
object A3Main extends App {
    val l1:List[Int] = List(1, 2, 3, 4, 2, 3)
    val l2:List[Int] = List(2, 1, 3, 4, 1)
    val l3:List[Int] = List(5, 6, 0, 9, 7)
    val l4:List[String] = List("one", "two")
    val l5:List[String] = List("two", "three")
    val l6:List[String] = List("four", "five")

    // test 1
    val b1:Boolean = A3.test(l1,l2)
    println(b1 + " [corretto: " + false + "]")

    // test 2
    val b2:Boolean = A3.test(l1,l3)
    println(b2 + " [corretto: " + true + "]")

    // test 3
    val b3:Boolean = A3.test(l4,l5)
    println(b3 + " [corretto: " + false + "]")

    // test 4
    val b4:Boolean = A3.test(l4,l6)
    println(b4 + " [corretto: " + true + "]")

    // test 5
    val b5:Boolean = A3.test(l1,l6)
    println(b5 + " [corretto: " + true + "]")
}
```

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