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

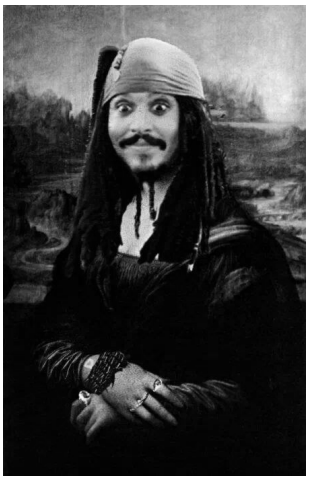
B

Esame del 23/01/2017 – Durata 1h 30' (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 $l \times l$, dove l è il minimo tra w e h, come nell'esempio sotto.



(a) Immagine originale

(b) Immagine output

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

```
void reflect(unsigned char* in, unsigned w, unsigned h,
             unsigned char** out, unsigned* ol,
             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 `l*l*sizeof(unsigned char)` byte che deve contenere l'immagine di output in formato row-major; **il buffer deve essere allocato nella funzione `mirror`**;
- `ol`: puntatore a `unsigned` in cui scrivere il lato l dell'immagine di output in pixel.

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

---

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

Esercizio 2 (Sottosequenza più lunga con una data proprietà)

Scrivere una funzione generica `maxSubSeq` che, data una lista di elementi generici e un predicato (cioè una funzione che restituisce un Boolean), restituisce una lista che contiene la la più lunga sottosequenza di elementi consecutivi della lista che soddisfa il predicato (si veda il programma di prova sotto per degli esempi).

Scrivere la soluzione in un file `B2.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `B2Main.scala`:

```
object B2Main extends App {
    // test 1: trova sottosequenza più lunga positiva
    val l1:List[Int] = List(9, 4, -1, -3, 7, 3, 6, -1)
    val m1:List[Int] = B2.maxSubSeq(l1, (x:Int) => x > 0);
    println(m1 + " [corretto: " + List(7,3,6) + "]")

    // test 2: trova sottosequenza più lunga di stringhe con < 3 caratteri
    val l2:List[String] = List("force", "may", "the", "jedi", "obi")
    val m2:List[String] = B2.maxSubSeq(l2, (x:String) => x.length <= 3);
    println(m2 + " [corretto: " + List("may", "the") + "]")
}
```

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

Esercizio 3 (Verifica se due liste contengono gli stessi elementi)

Scrivere una funzione generica che, date due liste, verifica se contengono insiemisticamente gli stessi elementi.

Scrivere la soluzione in un file `B3.scala` in modo che sia possibile compilare ed eseguire correttamente il seguente programma di prova `B3Main.scala`:

```
object B3Main 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(2, 1, 3, 1, 9)

    // test 1
    val b1:Boolean = B3.test(l1,l2)
    println(b1 + " [corretto: " + true + "]")

    // test 2
    val b2:Boolean = B3.test(l1,l3)
    println(b2 + " [corretto: " + false + "]")
}
```

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