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

A

Esame del 18/02/2019 – Durata 1h 45'

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

Esercizio 1 (Scala)

Si vuole rendere possibile invocare su una lista di elementi generici `T` un metodo `def getDup:Set[T]` che restituisce l'insieme degli elementi della lista che appaiono più di una volta. Ad esempio: `List(9,2,3,9,2,9).getDup` deve restituire `Set(2,9)` oppure `Set(9,2)`, che denotano lo stesso insieme.

Scrivere la soluzione in modo che sia possibile compilare ed eseguire correttamente il programma di prova `E1/E1Main.scala` riportato insieme al compito. La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare alcuna variabile `var`. Non toccare alcun file tranne `E1.scala`. Usare `make` per compilare e `make clean` per eliminare i `.class` prima di consegnare il compito.

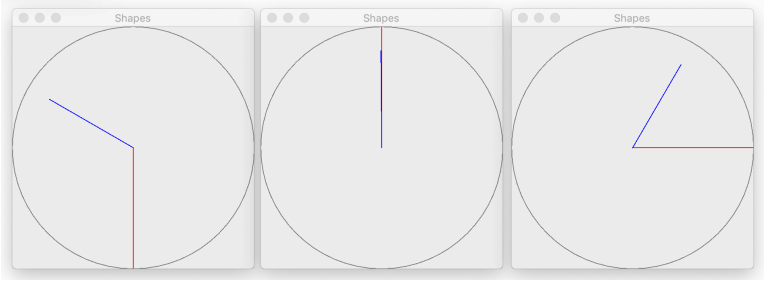
Esercizio 2 (Scala)

Si vuole scrivere nel file `E2/E2.scala` un metodo Scala `def getClock(hour:Int, min:Int):List[Shape]` che modella un semplice orologio analogico. Lo spazio delle coordinate della finestra è dato dai valori x e y `Double` compresi tra [0,1], dove (0,0) è la coordinata del punto nell'angolo inferiore sinistro e (1,0) è la coordinata del punto nell'angolo inferiore destro.

L'orologio deve essere costruito come un modello 2D formato da una lista di tre oggetti `Shape` (si veda il file `E2/Frame2D.scala`), nel seguente ordine:

1. **Quadrante**: cerchio di colore `Color.GRAY` inscritto nella finestra, assunta quadrata.
2. **Lancetta dei minuti**: segmento di linea di colore `Color.RED` e lunghezza $l_m = 0.5$. L'estremità mobile della lancetta ha coordinate $(0.5 + l_m \cdot \cos\mu, 0.5 + l_m \cdot \sin\mu)$, dove $\mu = \frac{\pi}{2} - 2\pi \cdot \frac{min}{60}$.
3. **Lancetta delle ore**: segmento di linea di colore `Color.BLUE` e lunghezza $l_h = 0.4$. L'estremità mobile della lancetta ha coordinate $(0.5 + l_h \cdot \cos\omega, 0.5 + l_h \cdot \sin\omega)$, dove $\omega = \frac{\pi}{2} - 2\pi \cdot \frac{hour}{12}$.

Si ricordi che la costante π è approssimata da `Math.PI` e che seno e coseno sono dati dai metodi `Math.sin` e `Math.cos`, rispettivamente. Scrivere la soluzione in un in modo che sia possibile compilare ed eseguire correttamente il programma di prova `E2/E2Main.scala` riportato insieme al compito, che presenta le seguenti tre finestre (sovrapposte) e il **risultato dei test sul terminale**:



La soluzione non deve usare alcun costrutto della programmazione imperativa e in particolare

alcuna variabile `var`. Non toccare alcun file tranne `E2.scala`. Usare `make` per compilare e `make clean` per eliminare i `.class` prima di consegnare il compito.

Esercizio 3 (Vettorizzazione SSE)

Si vuole realizzare una variante vettorizzata mediante SSE della seguente funzione C fornita nel file `E3/sumeven.c` che calcola la somma degli elementi di indici pari di un array di `n` `int`:

```
int sumeven(const int* v, int n) {
    int i, s = 0;
    for (i=0; i<n; i+=2) s += v[i];
    return s;
}
```

La soluzione deve essere scritta nel file `E3/sumeven_sse.c`. Compilare il programma di prova con `make` ed eseguirlo con `./sumeven`. Non toccare nessun file tranne `sumeven_sse.c`. Eliminare l'eseguibile con `make clean` prima di consegnare il compito.

Suggerimento. È possibile usare il seguente intrinsic SSE:

- `__m128i _mm_setzero_si128()`: restituisce un packed 128-bit integer con tutti i 128 bit inizializzati a zero.

Esercizio 4 (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 $w \times h$ con w e h pari, crei una nuova immagine delle stesse dimensioni ottenuta replicando ogni pixel di input con entrambi le coordinate (x,y) pari nelle posizioni di output (x,y), (x+1,y), (x,y+1), (x+1,y+1). Esempio:

(a) Immagine di input

(b) Immagine di output

Si completi nel file `E4/px4.c` la funzione `px4` con il seguente prototipo:

```
void px4(unsigned char* in, unsigned char* out, int h, int w,
         double* t, clut_device* dev)
```

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];
- `out`: puntatore a un buffer di dimensione `w*h*sizeof(unsigned char)` byte che contiene l'immagine di output in formato row-major;
- `h`: altezza di `in` e `out` in pixel (numero di righe della matrice di pixel);
- `w`: larghezza di `in` e `out` in pixel (numero di colonne della matrice di pixel);

---

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

- `t`: puntatore a `double` in cui scrivere il tempo di esecuzione in secondi richiesto dall'esecuzione del kernel.

Per compilare usare il comando `make`. Lanciare il programma con `./px4`. Verrà prodotta l'immagine di output `results/colosseo-px4.pgm`. Non toccare nessun file tranne `px4.c` e `px4.cl`.

Nota: è normale che in un ambiente virtualizzato come VirtualBox con immagini di piccole dimensioni il tempo di esecuzione della versione sequenziale sia inferiore a quello della versione OpenCL.