Esercitazione riassuntiva della prima parte del corso di Metodologie di Programmazione

canale P-Z

Esercizio 1. Si consideri il seguente metodo Java:

```
public int m(int n) {
      int i = 0, app = 0;
      while (i < n) {
            app += i;
            i += 2;
      }
      return app;
}
```

1. Se ne scrivano le opportune precondizioni e postcondizioni;
2. si scriva inoltre un'invariante di ciclo utilizzabile per dimostrare che il metodo restituisce la somma dei primi n/2 numeri naturali pari
3. si dimostri formalmente la correttezza di tale metodo
4. si definisca una funzione di terminazione per il ciclo presente nel metodo
5. si dimostri formalmente che la funzione definita è realmente una funzione di terminazione.

(N.B.: in un compito scritto saranno presenti al più due dei punti sopra elencati. Esempi tipici potrebbero essere 1 e 2; 4 e 5; 3, dove però pre-, post-condizioni e invarianti sono date).

Esercizio 2. Si considerino le seguenti due specifiche per un metodo Java che calcola il fattoriale di un numero intero:

SPEC_1:

```
// requires: n ≥ 0
// modifies:
// throws:
// effects:
// returns: il fattoriale di n
```

SPEC_2:

```
// requires:
// modifies:
// throws: NegativeInputArgument
// effects:
// returns: il fattoriale di n
```

Si dica quale è più forte (cioè, vincola di più l'implementatore) o se le due specifiche non sono confrontabili. Si utilizzi a piacere uno dei tre metodi visti a lezione (confronto intuitivo, tramite formule logiche, tramite relazioni di transizione).

(N.B.: nel compito potrebbe venir richiesto un metodo specifico; quindi, bisogna saperli utilizzare tutti e tre!)

Esercizio 3. Si consideri la specifica SPEC_2 vista nell'Esercizio 2 e il seguente metodo Java per il calcolo del fattoriale di un numero intero:

```
public int f(int n) {
      if (n == 0) return 1; else return n*f(n-1);
}
```

Il metodo soddisfa la specifica? Perché? Cosa si può dire di SPEC_1?

Esercizio 4. Si progetti un ADT che realizzi le *code con priorità*. Tali code sono strutture dati che memorizzano elementi associati a una priorità (presa da un insieme totalmente ordinato, ad esempio gli interi). Le operazioni fondamentali sono:

- l'inserimento in coda di un nuovo elemento con l'associata priorità;
- l'estrazione dell'elemento di priorità massima; a parità di priorità vale la legge FIFO.

Esercizio 5. Si implementi in Java l'ADT definito nell'Esercizio 4.

(N.B.: nell'esercizio d'esame l'ADT può essere dato o no; nel secondo caso, verrà richiesta la sua definizione, come nell'Esercizio 4)

Esercizio 6. Si consideri la seguente implementazione delle code con priorità, che usano interi sia per esprimere le priorità che gli elementi memorizzati:

```
class PriQueue {
      private int[][] V;
      // la prima riga della matrice contiene gli elementi
      // memorizzati, la seconda le priorità associate
      private int size, first, last;

      public PriQueue(int s){
            V = new int[s][2];
            size = s;
            first = last = 0;
      }

      public void ins(int elem, int pri) throws FullQueue{
            if (++last == size) throw FullQueue;
            V[last][0] = elem;
            V[last][1] = pri;
      }

      public int del() throws EmptyQueue{
            if (last == 0) throw EmptyQueue;
            first++;
            returns V[first-1][0];
      }
}
```

1. E' un'implementazione che soddisfa le specifiche poste dall'ADT sul metodo di cancellazione? Perché? Se l'implementazione non è corretta, si riscriva il metodo `del` (e non tutta la classe!) in modo che soddisfi l'ADT.

2. Si possono fare altre critiche all'implementazione scelta? Quali? Si proponga un'implementazione migliore, sia usando gli array che usando strutture dinamiche (liste).

(N.B.: anche qui, in un esame solo uno dei due punti precedenti verrà richiesto)