# 1° Esonero del corso di Metodologie di Programmazione

Canale P-Z

**Esercizio 1 (3+7 punti).** Si consideri il seguente ciclo:

```
while (i <= n) {// INV: i > 2 AND n > 2
      n += 2;
      i *= 2;
}
```

Si definisca una funzione di terminazione per il ciclo e si dimostri che la funzione definita è realmente una funzione di terminazione.

**Esercizio 2 (2+4+4 punti).** Si considerino le seguenti due specifiche per un metodo Java che calcola la radice quadrata di un numero reale:

S1:
```
//requires:
//modifies:
//throws: Negative Input Argument
//effects:
//returns: +√r
```

S2:
```
//requires: r ≥ 0
//modifies:
//throws:
//effects:
//returns: +√r OR -√r
```

Si dica quale è più forte (cioè, vincola di più l'implementatore) o se le due specifiche non sono confrontabili. Sia dia la risposta usando tutti e tre i metodi visti a lezione: confronto intuitivo, tramite formule logiche, tramite relazioni di transizione.

**Esercizio 3 (4+8 punti).** Si progetti un ADT che realizzi polinomi a coefficienti interi. Le operazioni fondamentali sono:

- la creazione di un polinomio di grado e coefficienti arbitrari;
- la somma e il prodotto di due polinomi.

Si implementi in Java l'ADT così definito.

# SOLUZIONI:

**Esercizio 1.** Una funzione di terminazione è $n\text{-}i$. Tale funzione è non-negativa ogni volta che si entra nel ciclo: infatti, se si è entrati nel ciclo, la guardia è vera e quindi $i \leq n$, da cui $n\text{-}i \geq 0$. Inoltre, la funzione decresce ad ogni iterazione: $(n\text{-}i)_{k+1} = n_{k+1} - i_{k+1} = n_k + 2 - 2i_k < n_k - i_k = (n - i)_k$, dove $i_k < 2i_k$, visto che $i_k > 2$ (per l'invariante).

**Esercizio 2.** La specifica S1 è più forte di S2. Ciò può essere dimostrato nei seguenti tre modi:

1. *Confronto intuitivo:* S1 richiede di meno all'utente (mentre S2 impone che l'argomento sia non-negativo), quindi l'implementatore può fare meno assunzioni sull'input; S1 impone all'implementatore di sollevare un'eccezione se l'input è negativo (mentre S2 lascia libero l'implementatore di gestire questa evenienza come vuole); S1 impone all'implementatore di restituire la radice quadrata positiva (mentre S2 lascia libero l'implementatore di scegliere se restituire la positiva o la negativa).
2. *Formule logiche:* trasformiamo S1 e S2 in formule logiche:
   F1 = true $\Rightarrow$ (throws (N.I.A.) $\wedge$ output(+√r))
   F2 = r $\geq 0 \Rightarrow$ (output(+√r) $\vee$ output(-√r))

Mostriamo che F1 $\Rightarrow$ F2 (mentre il viceversa non vale); ciò implica che S1 è più forte di S2.

| throws (N.I.A.) | output(+√r) | output(-√r) | r ≥ 0 | F1 | out(+√r)∨out(-√r) | F2 | F1 ⇒ F2 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

3. *Relazioni di transizione:* scriviamo le due relazioni di transizione:
   T1 = {(0,0), (1, +√1), (2,+√2), (3,+√3),…, (-1,N.I.A.), (-2,N.I.A.),…}
   T2 = {(0,0), (1, +√1), (1, -√1), (2,+√2), (2,-√2),…}
   Sembrerebbe quindi che né T1 $\subseteq$ T2 (visto che le coppie (-r,N.I.A.) $\in$ T1 ma $\notin$ T2), né T2 $\subseteq$ T1 (visto che le coppie (r,-√r) $\in$ T2 ma $\notin$ T1). Va però notato che T2 non è una relazione totale, mentre T1 lo è: quindi non si possono confrontare direttamente ma, come visto a lezione, si deve procedere in uno dei due possibili modi seguenti:
   - *restringere T1 al dominio di T2:* il dominio di T2 consiste di tutti i reali non-negativi; quindi T1 ristretta a tale dominio è {(0,0), (1, +√1), (2,+√2), (3,+√3),… }, che è contenuta in T2.
   - *completare T2:* in questo caso si devono aggiungere a T2 tutti i possibili comportamenti che l'implementatore può adottare nel caso in cui l'input non rispetti le precondizioni (cioè, sia negativo). Bisogna quindi aggiungere tutte le coppie del tipo
     - (-r, eccezione) per ogni possibile eccezione,
     - (-r, r') per ogni possibile reale r', e
     - (-r, NonTerminazione).

     Anche in questo caso si vede che, essendo N.I.A. una possibile eccezione, T1 $\subseteq$ T2.

**Esercizio 3.** L'ADT è il seguente:

Poly: int × int$^*$ → Poly
//PREC: l'input deve avere il formato (n, [$c_0$,…,$c_n$]),
// cioè se il grado è n bisogna passare n+1 coefficienti
//POSTC: restituisce un polinomio di grado n con $c_i$ come coefficiente i-esimo

sum: Poly × Poly → Poly
//POSTC: restituisce un polinomio di grado pari al grado massimo tra i due
//polinomi di input ottenuto sommando tutti i coefficienti di stesso grado

prod: Poly × Poly → Poly
//POSTC: restituisce un polinomio di grado pari alla somma dei gradi dei due
// polinomi di input ottenuto sommando tutti i prodotti tra coefficienti di stessa
// somma dei gradi

Una possibile implementazione in Java è la seguente:

```
class Poly {
      private int grado;
      private int[] coeff;

      public Poly(int g, int[] c){//PREC: c.length = g+1
            grado = g;
            coeff = c;
      }

      public Poly sum(Poly p){
            int gMax,gMin,i;
            int[] c;
            if (grado > p.gr()) then {
                  gMax = grado;
                  gMin = p.gr();
            }
            else {
                  gMax = p.gr();
                  gMin = grado;
            }
            c = new int[g];
            for (i = 0; i <= gMin; i++) c[i] = coeff[i]+p.co(i);
            for (; i <= gMax; i++){
                  if (gMax = grado) c[i] = coeff[i]; else c[i] = p.co(i);
            }
            return new Poly(gMax,c);
      }

      public Poly prod(Poly p){
            int[] c = new int[grado+p.gr()];
            for (int i = 0; i <= grado; i++)
                  for (int j = 0; j <= p.gr(); j++)
                        c[i+j] += coeff[i]*p.co(j);
            return new Poly(grado+p.gr(),c);
      }

      public int gr(){        // metodo di appoggio
            return grado;
      }

      public int co(int i){   // metodo di appoggio; PREC: i <= grado
            return coeff[i];
      }

}
```