Esercizio A

1. Definire la funzione `twice` che, presa una funzione f, la applica al valore 1 e al risultato così ottenuto applica di nuovo f.
2. Quale è il tipo di `twice`?
3. Applicare `twice` alla funzione anonima che incrementa di 1 valori interi.
4. Applicare `twice` alla funzione anonima che moltiplica per 10 valori interi.

Esercizio B

1. Definire la funzione `scalar : int -> int * int -> int * int` che presi n e (x,y) restituisce (n*x,n*y).
   Esempio:
   ```
   assert (scalar 3 (2,3)=(6, 9))
   ```

   Usare questa funzione per definire la funzione `doubleVec : int * int -> int * int` che raddoppia il vettore in input.
2. Definire la funzione `add_vect : int * int -> int * int -> int * int` che presi (x1,y1) e (x2,y2) restituisce (x1+x2,y1+y2).
   Esempio:
   ```
   assert(add_vect (1,2) (3,4)=(4, 6))
   ```

   Usa questa funzione per definire le funzioni `moveRight : int * int -> int * int` e `moveUp : int * int -> int * int` che prendono in input un intero n e un vettore (x,y) e traslano quest'ultimo di n unità rispettivamente verso destra e verso l'alto.
3. Definire la funzione `scalar_prod : int * int -> int * int -> int` che presi (x1,y1) e (x2,y2) restituisce x1*x2+y1*y2.
   Esempio:
   ```
   assert(scalar_prod (1,2) (3,4)=11)
   ```

   Usa questa funzione per definire le funzioni `sumVec : int * int -> int` e `diffVec : int * int -> int` che calcolano rispettivamente la somma e la differenza delle componenti di un vettore. Usa poi una di queste due funzioni per definire la funzione `isDiagonal : int * int -> bool` che controlla se un vettore sta sulla bisettrice del primo e terzo quadrante.
4. Definisci le funzioni ai punti precedenti nelle loro versioni uncurried.

Esercizio C

Nota: per definire funzioni ricorsive bisogna aggiungre la keyword `rec`, ad esempio `let rec f = ...`.

1. Definire la funzione generica `genSum : (int -> int) -> int -> int` tale che `genSum f n` calcola `f 0 + f 1 + ... + f n`.

   Una funzione è una specializzazione di `genSum` se ottenuta chiamando `genSum` e passando un'opportuna funzione come primo argomento.

   Definire come specializzazioni di `genSum` le funzioni `sumSquare` e `sumCube` che calcolano la somma dei quadrati e cubi dei numeri naturali da 0 a n inclusi.

   ```
   assert(sumSquare 3=14)

   assert(sumCube 3=36)
   ```

2. Definire la funzione generica `genProd : (int -> int) -> int -> int` tale che `genProd f n` calcola `f 0 * f 1 * ... * f n`.

   Definire come specializzazioni di `genProd` le funzioni `fact` e `twoRaisedTo` che calcolano il fattoriale di n e 2 elevato alla n.

   ```
   assert(fact 5=120)

   assert(twoRaisedTo 10=1024)
   ```

Esercizio D (difficile)

Considera la seguente definizione di funzione

```
let mapCollect (f1,f2) g (x,y) = g (f1 x) (f2 y)
```

Definisci le funzioni dell'esercizio B come specializzazioni di `mapCollect`.