Esercizi proposti – 3

1. Rappresentiamo le ore della giornata mediante coppie `(h,m): int * int`, dove `h` è compreso tra 0 e 23, inclusi (le ore) e `m` è compreso tra 0 e 59, inclusi (i minuti). Scrivere un programma con una funzione

   `somma_ore: (int * int) -> (int * int) -> int * int`,

   che calcoli la somma di due ore così rappresentate.

   Ad esempio:

   ```
   somma_ore (3,15) (4,20) = (7,35)
   somma_ore (3,45) (4,20) = (8,5)
   somma_ore (23,45)(0,20) = (0,5)
   ```

   Se uno dei due argomenti non è la rappresentazione corretta di un'ora, la funzione solleverà un'eccezione.

   Ricordarsi, anche in questo esercizio, le funzioni predefinite di OCaml per calcolare il modulo e la divisione intera.

2. Definire funzioni che risolvano i problemi seguenti (le funzioni avranno il nome e il tipo indicato all'inizio):

   (a) (`read_max: unit -> int`) Leggere da tastiera una sequenza di numeri interi (anche negativi), separati da Enter e terminata dalla stringa vuota (o da una qualsiasi stringa che non rappresenti un intero), e calcolarne il massimo. Se non viene immesso alcun numero, la funzione solleverà un'eccezione.

   (Tenere presente la funzione predefinita `max: 'a -> 'a -> 'a`).

   (b) (`read_max_min: unit -> int * int`) Leggere da tastiera una sequenza di numeri interi (anche negativi), separati da Enter e terminata dalla stringa vuota (o da una qualsiasi stringa che non rappresenti un intero), e calcolarne il massimo e il minimo. Se non viene immesso alcun numero, la funzione solleverà un'eccezione.

   (Tenere presente le funzione predefinite `max: 'a -> 'a -> 'a` e `min: 'a -> 'a -> 'a`).

   (c) (`tutti_minori: int -> bool`) Dato un intero n, leggere da tastiera una sequenza di interi (anche negativi), separati da Enter e terminata dalla stringa vuota (o da una qualsiasi stringa che non rappresenti un intero), e determinare (riportando `true` o `false`) se i numeri letti sono tutti minori di n. La funzione non deve mai sollevare eccezioni e la sua esecuzione non deve terminare finché non viene immessa la stringa vuota (o non numerica).

   (d) (`occorre: int -> bool`) Dato un intero n, leggere da tastiera una sequenza di interi, separati da Enter e terminata dalla stringa vuota (o da una qualsiasi stringa non numerica), e determinare (riportando `true` o `false`) se n occorre nella sequenza. La funzione non deve mai sollevare eccezioni e la sua esecuzione non deve terminare finché non viene immessa la stringa vuota (o non numerica).

   (e) (`num_di_stringhe: unit -> int`) Leggere da tastiera una sequenza di stringhe non vuote, separate da Enter e terminata dalla stringa

vuota, e riportare la lunghezza della sequenza (il numero di stringhe immesse, esclusa la stringa vuota che termina la sequenza). La funzione non deve mai sollevare eccezioni.

(f) (`stringa_max: unit -> string`) Leggere da tastiera una sequenza di stringhe non vuote, separate da Enter e terminata dalla stringa vuota, e riportare la stringa di lunghezza massima. Se non viene immessa nessuna stringa non vuota, la funzione riporterà la stringa vuota.

(Tenere presente la funzione predefinita `String.length: string -> int`).

3. Definire:

(a) `sumbetween: int -> int -> int`, tale che `sumbetween n m` = somma degli interi compresi tra n e m (estremi inclusi).

(b) `sumto: int -> int`, tale che `sumto n` = somma degli interi compresi tra 0 e n (incluso), assumendo $n \geq 0$.

(c) `power: int -> int -> int`, tale che `power n k` = k-esima potenza di n (assumendo $n, k \geq 0$).

(d) `fib: int -> int`, tale che `fib n` = n-esimo numero di Fibonacci (assumendo $n \geq 0$).

La sequenza dei numeri di Fibonacci è così definita:

```
fib 0 = 0,
fib 1 = 1,
fib n = fib (n-1) + fib(n-2)
```

(e) `maxstring: string -> char`, tale che `maxstring s` = massimo carattere in s (secondo il codice ASCII).

Ad esempio: `maxstring "antonio"= 't'`. Se l'argomento è la stringa vuota, la funzione solleverà un'eccezione.