PROGRAMMAZIONE II (A,B) - a.a. 2015-16
Primo Appello — Gennaio 2016

Esercizio 1

Si consideri il tipo di dati astratto modificabile `OrderedList<E>` che rappresenta una collezione di elementi omogenei tra loro ordinati in modo non decrescente. Supponiamo che la definizione del tipo di dato `OrderedList<E>` contenga tra gli altri i metodi

```
/* inserisce l'elemento elem nella lista rispettando l'ordine */
public void insert(E elem);

/* restituisce ed elimina dalla collezione l'elemento minimo dell'ordine */
public E extractMin();
```

1. Assumendo di adottare una strategia di programmazione difensiva, si completi la specifica dei metodi `insert` e `extractMin`, definendo le clausole REQUIRES, MODIFIES e EFFECTS, indicando le eccezioni eventualmente lanciate e se sono checked o unchecked.

   Si veda il file `OrderedList.java`.

2. Si consideri la seguente struttura di implementazione per la classe `OrderedList<E>`

   ```
   private ArrayList<E> elems;
   private ArrayList<Integer> ordine;
   ```

   Intuitivamente, `elems` contiene tutti gli elementi della lista, in un ordine qualunque, mentre `ordine` contiene per ogni indice i, $0 \leq i \leq elems.size() - 1$, l'indice del successore in `elems` di `elems.get(i)`. Se l'elemento di indice i non ha un successore (ovvero, é l'elemento di valore massimo) allora il valore di `ordine` all'indice i risulta essere -1. Ad esempio, se `elems` contiene `[E A B F C]` allora `ordine` contiene `[3 2 4 -1 0]`.

   Si definisca l'invariante di rappresentazione per l'implementazione di `OrderedList<E>`.

   Si veda il file `OrderedListImpl.java`.

3. Si fornisca l'implementazione del metodo `insert` e si dimostri che preserva l'invariante di rappresentazione.

   Si veda il file `OrderedListImpl.java`.

Esercizio 2

Si estenda il linguaggio con il costrutto condizionale **on** G1 ; E1 — G2 ; E2 dove G1 e G2 sono espressioni booleane dette guardie. La valutazione del costrutto `on` avviene nel modo seguente

- si valutano le guardie;
- se nessuna della due guardie viene valutata a true si solleva una eccezione;

- se una sola delle espressione viene valutata a true, si valuta l'espressione associata;
- se entrambe l'espressioni vengono valutate a true, viene selezionato in modo casuale l'espressione da valutare.

Si mostri come deve essere modificato l'interprete del linguaggio didattico funzionale.
Sintassi

```
type exp = ...
   | On of exp * exp * exp * exp
```

Semantica

```
let rec sem e r = match e with
   ...
   | On(g1, e1, g2, e2) -> match (sem(g1, r), sem(g2, r)) with
                              (t, _) -> sem(e1, r)
                            | (f, t) -> sem(e2, r)
                            | (f, f) -> failwith "false guards"
                            | (_, _) -> failwith "wrong exp"
```

Esercizio 3

Si consideri il seguente programma scritto in un linguaggio imperativo che ammette definizioni globali di variabili e funzioni

```
int x = 1;
int n = 10;

function void f() {
   int x = 3;

   function void g(int c, int d) {
      c = c - 1;
      print(c, d, x, n);
   }

   g(x, x); /* body di  f */
}

function void h() {
   int n = 2;

  f();
}

main()  { h(); }
```

1. Assumendo scoping statico e passaggio per valore
   (a) si dica quali sono i valori stampati dal programma; *I quattro valori sono 2 3 3 10.*

(b) si descriva la struttura del run time stack nella valutazione del programma.

2. Assumendo scoping dinamico e passaggio per riferimento, si dica quali sono i valori stampati dal programma. *I quattro valori sono* `2 2 2 2`.

Esercizio 4

Si consideri il seguente frammento di codice Java

```
void start() {
  A a = new A();     /* Linea 1 */
  B b = new B();     /* Linea 2 */
  C c = new C();     /* Linea 3 */
  a.s(b, c);         /* Linea 4 */
  b = null;          /* Linea 5 */
  a = null;          /* Linea 6 */
  c = null;          /* Linea 7 */
}
```

Si indichi, motivando la risposta, quando l'oggetto `c` creato alla linea 3 diventa garbage.
Non è detto che lo diventi affatto, dato che potrebbe essere utilizzato nel corpo del metodo `s`.