

A spiral-bound notebook with a light brown, textured cover and a silver metal spiral binding on the left side. The notebook is open to a blank page with a light beige, textured paper. The text is centered on the page.

Un iteratore per le liste ordinate di interi

Nuova specifica di OrderedIntList

```
public class OrderedIntList {
    public OrderedIntList ()
    public void addEl (int el) throws
        DuplicateException
    public void remEl (int el) throws
        NotFoundException
    public boolean isIn (int el)
    public boolean isEmpty ()
    public int least () throws EmptyException
    public Iterator smallToBig ()
        // EFFECTS: ritorna un generatore che produrrà gli
        // elementi di this (come Integers), in ordine
        // crescente
        // REQUIRES: this non deve essere modificato finché
        // il generatore è in uso
    public boolean repOk ()
    public String toString ()}
```

- ✓ senza l'iteratore, non ci sarebbe nessuna operazione per accedere gli elementi

Il generatore

```
public class OrderedIntList {
// OVERVIEW: una OrderedIntList è una lista
// modificabile di interi ordinata
// tipico elemento: [x1, ..., xn], xi < xj se i < j
private boolean vuota;
private OrderedIntList prima, dopo;
private int val;
public Iterator smallToBig ()
// EFFECTS: ritorna un generatore che produrrà gli
// elementi di this (come Integers), in ordine
// crescente
// REQUIRES: this non deve essere modificato finché
// il generatore è in uso
{return new OGen(this, count());}

private int count () {
if (vuota) return 0;
return 1 + prima.count() + dopo.count(); }
✓ al generatore viene passato il numero di elementi della lista
– calcolato dal metodo privato count
```

Implementazione del generatore 1

```
public class OrderedIntList {
// OVERVIEW: una OrderedIntList è una lista
// modificabile di interi ordinata
// tipico elemento: [x1, ..., xn], xi < xj se i < j
private boolean vuota;
private OrderedIntList prima, dopo;
private int val;
private static class OGen implements Iterator {
private int cnt; // numero di elementi ancora da generare
private OGen figlio; // sottogeneratore corrente
private OrderedIntList me; // il mio nodo
// la funzione di astrazione (ricorsiva!)
//  $\alpha(c)$  = se  $c.cnt = 0$  allora [], altrimenti
// se il numero di elementi di  $\alpha(c.figlio)$  è  $c.cnt$ 
// allora  $\alpha(c.figlio)$ ,
// altrimenti  $\alpha(c.figlio) + [Integer(c.me.val)] +$ 
//  $\alpha(OGen(c.dopo))$ 
}
```

Implementazione del generatore 2

```
public class OrderedIntList {
// OVERVIEW: una OrderedIntList è una lista
// modificabile di interi ordinata
// tipico elemento: [x1, ..., xn], xi < xj se i < j
private boolean vuota;
private OrderedIntList prima, dopo;
private int val;
private static class OGen implements Iterator {
private int cnt; // numero di elementi ancora da generare
private OGen figlio; // sottogeneratore corrente
private OrderedIntList me; // il mio nodo
// l'invariante di rappresentazione
// l(c) = c.cnt = 0 oppure
// (c.cnt > 0 e c.me != null e c.figlio != null e
// (c.cnt = c.figlio.cnt + 1 oppure
// c.cnt = c.figlio.cnt + c.me.dopo.count() + 1))
}
```

Implementazione del generatore 3

```
public class OrderedIntList {
// OVERVIEW: una OrderedIntList è una lista
// modificabile di interi ordinata
// tipico elemento: [x1, ..., xn], xi < xj se i < j
private boolean vuota;
private OrderedIntList prima, dopo;
private int val;
private static class OGen implements Iterator {
private int cnt; // numero di elementi ancora da generare
private OGen figlio; // sottogeneratore corrente
private OrderedIntList me; // il mio nodo

    OGen (OrderedIntList o, int n) {
        // REQUIRES: o != null
        cnt = n;
        if (cnt > 0) { me = o;
            figlio = new OGen(o.prima, o.prima.count()); }
    }
}
```

✓ anche il costruttore è ricorsivo

Implementazione del generatore 3.1

```
public class OrderedIntList {
    private boolean vuota;
    private OrderedIntList prima, dopo;
    private int val;
    private static class OGen implements Iterator {
        private int cnt; // numero di elementi ancora da generare
        private OGen figlio; // sottogeneratore corrente
        private OrderedIntList me; // il mio nodo
        // l(c) = c.cnt = 0 oppure
        // (c.cnt > 0 e c.me != null e c.figlio != null e
        // (c.cnt = c.figlio.cnt + 1 oppure
        // c.cnt = c.figlio.cnt + c.me.dopo.count() + 1))
        OGen (OrderedIntList o, int n) {
            // REQUIRES: o != null
            cnt = n;
            if (cnt > 0) { me = o;
                figlio = new OGen(o.prima, o.prima.count()); }
        }
    }
    ✓ cnt = 0 oppure
    ✓ cnt > 0 e me != null (clausola REQUIRES) e
    ✓ figlio != null (chiamata ricorsiva) e
    ✓ disgiunzione sui count (proprietà di count)
```

Implementazione del generatore 4

```
public class OrderedIntList {  
    // OVERVIEW: una OrderedIntList è una lista  
    // modificabile di interi ordinata  
    // tipico elemento: [x1, ..., xn], xi < xj se i < j  
    private boolean vuota;  
    private OrderedIntList prima, dopo;  
    private int val;  
    private static class OLGGen implements Iterator {  
        private int cnt; // numero di elementi ancora da generare  
        private OLGGen figlio; // sottogeneratore corrente  
        private OrderedIntList me; // il mio nodo  
  
        public boolean hasNext () {  
            return cnt > 0; }  
    }
```

- ✓ si noti l'uso di `cnt` per rendere efficiente anche questo metodo

Implementazione del generatore 5

```
public class OrderedIntList {
// OVERVIEW: una OrderedIntList è una lista
// modificabile di interi ordinata
// tipico elemento: [x1, ..., xn], xi < xj se i < j
private boolean vuota;
private OrderedIntList prima, dopo;
private int val;
private static class OGen implements Iterator {
private int cnt; // numero di elementi ancora da generare
private OGen figlio; // sottogeneratore corrente
private OrderedIntList me; // il mio nodo
public Object next () throws NoSuchElementException {
if (cnt == 0) throw new
    NoSuchElementException("OrderedIntList.smallToBig") ;
cnt --;
try { return figlio.next(); }
catch (NoSuchElementException e) {}
// se arriva qui ha finito tutti quelli prima
figlio = new OGen(me.dopo, cnt);
return new Integer(me.val); }
}
```