Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2011/2012

16 gennaio 2012

1. Provare che la seguente grammatica è ambigua.

   ```
   Exp ::= if Exp then Exp | if Exp then Exp else Exp | ( Exp ) | true | false
   ```

   Definire una grammatica non ambigua che generi lo stesso linguaggio.

2. (a) Definire, in modo diretto e senza parametro di accumulazione, la funzione

   ```
   delete : 'a -> 'a list -> 'a list
   ```

   che cancella un elemento da una lista ordinata **senza ripetizioni**.

   (b) Usando la funzione `delete` del punto precedente, e la funzione

   ```
   itlist : ('a -> 'b -> 'b) -> 'b -> 'a list -> 'b
   let rec itlist f a = function x::l -> itlist f (f x a) l | _ -> a;;
   ```

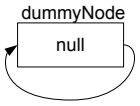
   definire la funzione

   ```
   diff : 'a list -> 'a list -> 'a list
   ```

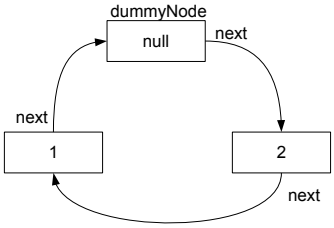
   tale che, per ogni lista ordinata senza ripetizioni l_1 ed l_2, `diff` l_1 l_2 restituisca la lista ordinata senza ripetizioni che contiene tutti e soli gli elementi di l_1 che non appartengono ad l_2.

   (c) Definire la funzione `diff` del punto precedente, ma in modo diretto e senza parametro di accumulazione.

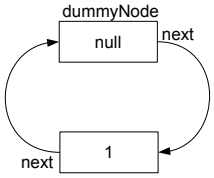
3. Completare la seguente classe `LinkedStack` che implementa stack generici (che possono contenere anche elementi **`null`**) tramite liste circolari con nodi collegati singolarmente (al nodo successivo) e con nodo fittizio (dummy node). Lo stack vuoto è costituito da un unico nodo fittizio che punta a se stesso, come suggerisce la seguente rappresentazione grafica:



Lo stack dove sono stati inseriti tramite il metodo `push`, nell'ordine, gli elementi 1 e 2, è costituito dai seguenti tre nodi:



Infine, invocando il metodo `pop`, si ottiene il seguente stack:



```java
package scritto2012_01_16;
public interface Stack<E> extends Iterable<E>{
	// Pushes item onto the top of this stack.  Returns the item argument
	public E push(E item);
	// Removes the object at the top of this stack and returns that object
	// Throws EmptyStackException if this stack is empty
	public E pop();
	// Returns the object at the top of this stack
	// Throws EmptyStackException if this stack is empty
	public E peek();
	// Returns true if and only if this stack contains no items; false otherwise
	public boolean empty();
}
```

```java
package scritto2012_01_16;
import java.util.Iterator;
import java.util.EmptyStackException;
import java.util.NoSuchElementException;
public class LinkedStack<E> implements Stack<E> {
	private final Node<E> dummyNode;
	private static class Node<E> {
		private E elem;
		private Node<E> next;
		private Node(E elem, Node<E> next) {
			this.elem = elem;
			this.next = next;
		}
	}
	@Override
	public String toString() {
		String res = "[";
		for (E e : this)
			res += " " + e;
		return res + " ]";
	}
	@Override
	public Iterator<E> iterator() {
		return new Iterator<E>() {
			private Node<E> prevNode = dummyNode;
			@Override
			public boolean hasNext() {
				return prevNode.next != dummyNode;
			}
			@Override
			public E next() {
				if (!hasNext())
					throw new NoSuchElementException();
				prevNode = prevNode.next;
				return prevNode.elem;
			}
			@Override
			public void remove() {
				throw new UnsupportedOperationException();
			}
		};
	}
	@Override
	public int hashCode() {
		int hashCode = 17;
		for (E e : this)
			hashCode = 31 * hashCode + ((e == null) ? 0 : e.hashCode());
		return hashCode;
	}
	// Creates an empty Stack
	public LinkedStack() { /* completare */ }
	// Returns true if and only if the specified Object is also a Stack, and
	// both stacks have the same size and contain equal elements in the same order
	@Override
	public boolean equals(Object obj) { // completare usando due iterator.
					/* deve funzionare anche con elementi null */ }
	@Override
	public E push(E item) { /* completare */ }
	@Override
	public E pop() { /* completare */ }
	@Override
	public E peek() { /* completare */ }
	@Override
	public boolean empty() { /* completare */ }
}
```

4. Considerare le seguenti dichiarazioni di classi Java, contenute nello stesso package:

```
class A {
    private String m(Number n){ return "A.Number"; }
    protected String m(Integer i){ return "A.Integer " + m(1.4); }
}
class B extends A {
    protected String m(Integer i){ return "B.Integer " + super.m(i); }
}
class C extends B {
    protected String m(Number... n){ return "C.Number"; }
}
class Main {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        ...
   }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Main` l'espressione indicata.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

(a) `out.println(a.m(1));`

(b) `out.println(a.m(1.4));`

(c) `out.println(b.m(1));`

(d) `out.println(((A) b).m(1));`

(e) `out.println(c.m(1));`

(f) `out.println(c.m(1.4));`