Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2011/2012

13 luglio 2012

1. Sia $\mathcal{L}$ il linguaggio generato dalla seguente grammatica a partire dal simbolo non terminale `Exp`.

```
Exp  ::= Exp = Exp | Id | Bit | ( Exp )
Id ::= x | y | z
Bit ::= 0 | 1
```

   (a) Dimostrare che la grammatica è ambigua.

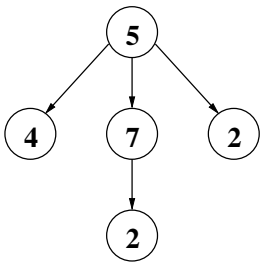
   (b) Definire una grammatica **non ambigua** che generi $\mathcal{L}$ in modo che l'operatore **=** sia **associativo da destra**.

2. Sia `'a tree` il seguente tipo user-defined

```
type 'a tree = Node of 'a * 'a tree list;;
```

   degli alberi non vuoti i cui nodi possono avere un numero arbitrario di figli e sono etichettati con valori di tipo `'a`.

   Per esempio, il termine `Node(5,[Node(4,[]);Node(7,[Node(2,[])]);Node(2,[])])` corrisponde all'albero



   (a) Definire in modo diretto la funzione `tree_member : 'a -> 'a tree -> bool` tale che `tree_member` x t restituisca `true` se e solo se l'albero t contiene un nodo etichettato con x.

   (b) Siano

```
tree_exists : ('a -> bool) -> 'a tree -> bool
forest_exists : ('a -> bool) -> 'a tree list -> bool
```

   le seguenti funzioni mutuamente ricorsive:

```
let rec tree_exists p = function
    Node(x,l) -> p x || forest_exists p l
and forest_exists p = function
    [] -> false
  | t::l -> tree_exists p t || forest_exists p l;;
```

   Definire la funzione `tree_member` specificata al punto precedente, come opportuna specializzazione della funzione `tree_exists`.

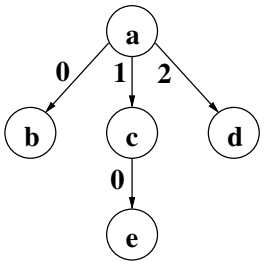
   (c) Definire la funzione `count_tree_nodes : 'a tree -> int` tale che `count_tree_nodes` t restituisca il numero di nodi di t.

3. Considerare le seguenti interfacce e classi:

   - `Tree<E>`: alberi non vuoti i cui nodi possono avere un numero arbitrario di figli e sono etichettati con valori di tipo `E`.
   - `Node<E>`: nodi dell'albero etichettati con valori di tipo `E`.
   - `TreeClass<E>`: implementazione di `Tree<E>`.

```
import java.util.Stack;

public interface Tree<E> {
        boolean contains(E el);
        E get(Stack<Integer> path);
        E set(Stack<Integer> path,E elem);
}

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

public class TreeClass<E> implements Tree<E> {
        private Node<E> root;

        private static class Node<E> {
                private E elem;
                private List<Node<E>> children;

                private Node(E elem, List<Node<E>> children) {
                        this.elem = elem;
                        this.children = children;
                }
                private Node(E elem) {
                        this(elem, new ArrayList<Node<E>>());
                }
                private boolean contains(E elem) {
                    // completare
                }
        }
        public TreeClass(E elem) {
            // completare
        }
        @Override
        public E get(Stack<Integer> path) {
            // completare
        }
        @Override
        public E set(Stack<Integer> path, E elem) {
            // completare
        }
        @Override
        public boolean contains(E elem) {
            // completare
        }
}
```

Nei metodi `get` e `set` ogni nodo viene individuato con il cammino (rappresentato da uno stack) dalla radice al nodo stesso, con la convenzione che il primo nodo nella lista `children` corrisponde all'indice 0. Per esempio, nell'albero



i nodi **a**, **b**, **c**, **d**, ed **e** sono rispettivamente individuati dai path `[]`, `[0]`, `[1]`, `[2]` e `[1,0]`.

(a) Completare la definizione del costruttore `TreeClass(E elem)` che crea un albero con la sola radice, etichettata da `elem`.

(b) Completare la definizione del metodo **`boolean`** `contains(E elem)` della classe `Node`, che restituisce **`true`** se e solo se l'albero la cui radice coincide con il nodo **`this`** contiene un nodo etichettato con `elem`.

(c) Completare la definizione del metodo **`boolean`** `contains(E elem)` della classe `TreeClass`, che restituisce **`true`** se e solo se l'albero **`this`** contiene un nodo etichettato con `elem`.

(d) Completare la definizione del metodo `E get(Stack<Integer> path)` che restituisce l'etichetta del nodo dell'albero **`this`** individuato dal cammino `path`.

(e) Completare la definizione del metodo `E set(Stack<Integer> path, E elem)` che associa all'etichetta del nodo dell'albero **`this`** individuato dal cammino `path` la nuova etichetta `elem`; il metodo restituisce il valore precedente dell'etichetta del nodo.

4. Considerare le seguenti dichiarazioni di classi Java:

```
package pck1;
public class C1 {
        void m() {
                System.out.println("C1.m");
        }
        public void q() {
                System.out.println("C1.q");
                m();
        }
}
-----------------------------------------
package pck1;
public class C2 extends C1 {
        void m() {
                System.out.println("C2.m");
        }
        public void q() {
                System.out.println("C2.q");
                super.q();
        }
}
-----------------------------------------
package pck2;
public class C3 extends pck1.C2 {
        protected void m(Object... objects) {
                System.out.println("C3.m");
        }
}
-----------------------------------------
package pck1;
public class C4 extends pck2.C3 {
        public void m() {
                super.m();
                System.out.println("C4.m");
        }
        protected void m(Object objects) {
                System.out.println("C3.m");
        }
        public void q() {
                System.out.println("C4.q");
                super.q();
        }
}
-----------------------------------------
package pck2;
import pck1.*;
public class Main {
    public static void main(String[] args) {
        C1 c1 = new C1();
        C2 c2 = new C2();
        C3 c3 = new C3();
        C4 c4 = new C4();
        ...
    }
}
```

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) `((C2) c3).m();`

(b) `c4.m();`

(c) `c3.q();`

(d) `((C3) c4).m();`

(e) `((C1) c2).q();`

(f) `c4.q();`