

PROGRAMMAZIONE II (A,B) - a.a. 2014-15

Prova scritta — 04 Settembre 2015

Esercizio 1. Si consideri il nucleo di un semplice linguaggio di programmazione funzionale, la cui sintassi è descritta di seguito

Pixel $p ::= \langle r, g, b \rangle,$
 $r, g, b \in \{0, 1, \dots, 255\}$

Identificatori $I ::= \dots$

Espressioni $e ::= I \mid p \mid \text{lighten } e \mid \text{darken } e \mid \text{let } I = e_1 \text{ in } e_2$

Intuitivamente, un *pixel* è un tipo di dato che contiene tre valori interi compresi tra 0 e 255 (il primo valore codifica *red*, il secondo *green* e il terzo *blue*). L'espressione “lighten” produce come risultato un pixel dove ogni componente del pixel passato come argomento viene incrementato di 1. L'incremento del valore 255 produce 255. L'espressione “darken” produce come risultato un pixel dove ogni componente del pixel passato come argomento viene diminuito di 1. Il decremento del valore 0 produce 0.

1. Si definisca l'interprete del linguaggio utilizzando OCaml come linguaggio di implementazione.

Esercizio 2. Si consideri il seguente programma OCaml

```
let x = v;;  
let f1 = fun z -> z*x;;  
let f2 = fun z -> let x = 2 in (f1 z) * x;;  
f2 4;;
```

1. Si dica per quale valore di v il programma precedente produce come risultato il valore 40. $v = 120$
2. Si descriva lo stato dello stack dei record di attivazione nella simulazione della valutazione del programma.
3. Assumendo di considerare una implementazione di OCaml basata su regole di scoping dinamico, si descriva lo stato dello stack dei record di attivazione nella simulazione della valutazione del programma.

Esercizio 3. Si consideri il seguente frammento di programma, in una sintassi Java-like. Il programma intende rappresentare un’astrazione per operare su liste di valori interi.

```
public class SLL {
    protected SLLNode head = null;

    public SLL() {}

    public void prepend(int x) {
        SLLNode n = new SLLNode(x, head);
        head = n;
    }
    public SLLNode getFirst() {
        return head;
    }
    public SLLNode findFirstGreaterThan(int x) {
        SLLNode n = head;
        while (n != null)
            if (n.getVal() > x) return n;
            else n = n.getNext();
        return null;
    }
}

public class SLLNode {
    private int value;
    protected SLLNode next = null;
    protected SLLNode(int x, SLLNode n) { value = x; next = n; }
    public int getVal() { return value; }
    public SLLNode getNext() { return next; }
}
```

Il nome della classe suggerisce intuitivamente che stiamo realizzando una *single-linked list*.

1. Definire per le classi `SLL` e `SLLNode` la specifica dei metodi e l’invariante di rappresentazione.
2. Si dimostri che l’implementazione del metodo `prepend` preserva l’invariante di rappresentazione.

Si considerino ora le classi

```
class DLL extends SLL {
    ...
    public DLL() { ... }
    public DLLNode getFirst() { ... }
    public DLLNode findFirstGreaterThan(int x) { ... }
}

class DLLNode extends SLLNode {
    private DLLNode prev;
    protected DLLNode(int x, DLLNode n) { ... }
    public DLLNode getNext() { ... }
    public DLLNode getPrev() { ... }
}
```

La classe `DLL` soddisfa il principio di sostituzione? In caso di risposta negativa se ne indichi il motivo.