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

$$\begin{array}{ll} \text{Pixel} & p ::= \langle r, g, b \rangle, \\ & r, g, b \in \{0, 1, \ldots, 255\} \\ \\ \text{Identificatori} & I ::= \ldots \\ \\ \text{Espressioni} & e ::= I \mid p \mid \mathit{lighten}\ e \mid \mathit{darken}\ e \mid \mathit{let}\ I = e_1\ \mathit{in}\ e_2 \end{array}$$

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.

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

```
type pixel = int * int * int

type eval = unbound | Epix of pixel

type ide = string

type exp = Ide of ide | Pix of pixel |
    | Light of exp | Dark of exp
    | Let of ide * exp * exp
```

```
let rec sem (e: exp) (r: eval env) = match e with
      Ide(i) -> r i
    | Pix (a, b, c) -> if check a & check b & check c then Epix(a, b, c) else unbound
    | Light e -> let v = sem(e, r) in increase v
    | Dark e -> let v = sem(e, r) in decrease v
    | Let (i, e1, e2) -> let v = sem(e1, r) in let r1 = bind(r, i, v) in sem(e2, r1)

let check (a: int) = if a > -1 & a < 256 then true else false

let increase (v: eval) = match v with
      Epix(a, b, c) -> Epix(inc a, inc b, inc c)

let inc (a: int) = match a with
      255 -> 255
    | n -> n + 1
```

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. 5
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;

   public SLL() { head = null; }

   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;

   public 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.

```
public class SLL {
   // Overview: Tipo modificabile di interi ordinati
   // Typical Element: [int1, int2, ..., intk]
   // FA = [head.getVal(), head.getNext().getVal(), ...,  head.getNext()^k.getVal()]
   //      se IR(k) con getNext()^k la ripetizione k volte di getNext()
   // IR = exists k. IR(k)
   // IR(k) = 0 <= k && head.getNext()^k.getNext() = null &&
   //         (forall i. 0 <= i < k ==> head.getNext()^i.getNext() != null)

   protected SLLNode head;

   /* crea una lista vuota */
   //@effects head==null
   public SLL() { head = null; }

   /* aggiunge un elemento in cima alla lista */
   //@effects modify this
   //@effects head==x
   public void prepend(int x) { ... }

   /* restituisce l'elemento in cima alla lista, o null se la lista e' vuota */
   //@effects result==head
   public SLLNode getFirst() { return head; }

   /* scandisce la lista dalla cima, restituendo il primo valore maggiore di x,
      o null altrimenti */
   //@effects result!=null ==> result>x
   public SLLNode findFirstGreaterThan(int x) { ... }
}
```

2. Si dimostri che l'implementazione del metodo `prepend` preserva l'invariante di rappresentazione.

```
Immediato: prepend aggiunge l'elemento in cima, e se prima vale IR(k), dopo IR(k+1).
```

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;
   public DLLNode(int x, DLLNode n) { ... }
   public DLLNode getNext() { ... }
   public DLLNode getPrev() { ... }
}
```

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