PROGRAMMAZIONE II (A,B) - a.a. 2013-14
Appello Straordinario — 6 Novembre 2014

Esercizio 1. Si consideri il seguente programma OCaml:

```
let f x y z =
    if z > 0 then (fun w -> w + x + y)
             else (fun w -> w + x - y);;
let a = 1;;
let b = 2;;
let c = f b a 7;;
let d = c 4;;
```

1. Indicare il tipo inferito dall'interprete OCaml per le variabili `f` e `c`.
2. Indicare il valore associato alla variabile `d` al termine dell'esecuzione del programma.
3. Simulando la valutazione del programma, mostrare la struttura della pila dei record di attivazione subito dopo l'invocazione di `let c = f b a 7`, e al momento dell'invocazione di `c 4`.

Esercizio 2. Si consideri un tipo di dato astratto *Membro* di supporto alla gestione di una rete sociale. La classe `Membro` ha, tra gli altri, i seguenti metodi:

```
public ArrayList<Membro> invitati();
// MODIFES: nothing (metodo osservatore)
// EFFECT: restituisce l'elenco dei membri invitati da this che non hanno ancora
// risposto all'invito.

public ArrayList<Membro> amici();
// MODIFIES: nothing (metodo osservatore)
// EFFECT: restituisce l'elenco dei membri invitati da this che hanno accettato, oppure
// che hanno invitato  this e di cui this ha accettato l'invito.

public void invita (Membro m);
// l'effetto  di invitare un altro membro m della rete sociale a diventare amico del membro
// this. Il metodo non deve permettere di invitare se stessi oppure un membro che ha
// gia' risposto positivamente a un invito

public void accettaInvito (Membro m);
// l'effetto e' quello di accettare l'invito di un altro membro m della rete sociale

public void elaboraNotifica (Membro m, boolean risposta);
// viene eseguito quando il membro m risponde all'invito di this: l'effetto dipende
// dalla risposta
```

1. Fornire l'OVERVIEW della classe, basandosi anche sui due metodi osservatori.
2. Fornire la specifica degli ultimi tre metodi descritti informalmente, indicando per ogni metodo:
   (a) le clausole REQUIRES, MODIFY ed EFFECT;
   (b) le eventuali eccezioni lanciate in dipendenza dei parametri attuali.

Si assuma che la classe `MembroImpl` implementi il tipo di dato astratto `Membro` con una struttura dati dichiarata nel modo seguente:

```
private ArrayList<Membro> altriMembri;
private int inizioInvitati;
public MembroImpl() {
  altriMembri = new ArrayList<Membro>;
  inizioInvitati = 0;
  }
```

L'idea è che un oggetto di tipo `MembroImpl` contiene un'unica lista di membri, in cui la parte iniziale contiene gli amici mentre quella a partire dalla posizione `inizioInvitati` contiene i membri con inviti pendenti.

3 Definire l'invariante di rappresentazione e la funzione di astrazione.

4 Implementare il metodo `elaboraNotifica`() verificando che preservi l'invariante di rappresentazione.

Esercizio 3. Si consideri il linguaggio didattico imperativo. Estendiamo il linguaggio in modo da includere la possibilità di dichiarare procedure in cui i parametri siano passati per *value-result*. Per semplicità supponiamo che le procedure abbiamo un unico parametro formale.

1. Estendere la sintassi astratta del linguaggio didattico imperativo in modo da includere la dichiarazione di procedure con parametri per value-result e la loro invocazione.
2. Definire le regole OCaml dell'interprete per trattare la valutazione di dichiarazione di procedure con parametri per value-result e la loro invocazione.