# PROGRAMMAZIONE II (A, B) - a.a. 2015-16
# Prova scritta – 7 giugno 2016

**Esercizio 1.** Si consideri il seguente programma OCaml

```
let add = fun x -> fun y -> x+y;;
let choice x y z =
  if z > 10 then add x
            else add y;;
let rec applyAll xs a b =
  match xs with
    [] -> []
  | hd::tl -> ((choice a b hd) hd)::(applyAll tl a b);;
let a = 1;;
let b = -1;;
let ls = [10; 20; 5];;
let res = applyAll ls a b;;
```

1. Indicare il tipo inferito dall'interprete OCaml per le variabili `choice` e `applyAll`.
2. Indicare il valore associato alla variabile `res` al termine dell'esecuzione del programma.
3. Simulare la struttura del supporto a run-time durante l'esecuzione del programma, mostrando come varia la struttura della pila dei record di attivazione.

**Esercizio 2.** Si consideri un tipo di dato astratto *Individual* di supporto alla gestione di una rete sociale. La classe `Individual<Profile>` è generica rispetto alla classe `Profile` che contiene tutti gli attributi che caratterizzano le informazione di un generico individuo, ad esempio `name`, `education`, `favouriteTeam` e così via, con gli opportuni metodi osservatori. Inoltre supponiamo che la classe `Individual` includa, tra gli altri, i seguenti metodi

- `public void insert(Profile p)` il cui effetto è quello di inserire un altro componente della rete sociale fra i contatti dell'individuo associato al profilo di `this`. Il metodo non deve permettere di inserire se stessi oppure un altro profilo già presente. Il metodo in base alle caratteristiche del parametro `p` inserisce l'individuo nella lista dei contatti pubblici o in quella dei contatti privati del profilo `this`.
- `public void remove(Profile p)` il cui effetto è quello di eliminare tra la lista dei contatti di `this` il profilo `p`.
- `public ArrayList<Profile> share()` il cui effetto è quello di condividere la propria lista dei contatti pubblici con un altro individuo della rete sociale.

1. Definire la specifica dei metodi descritti in precedenza e indicando per ogni metodo **a)** le clausole REQUIRES, MODIFY ed EFFECT, e **b)** il valore restituito e le eventuali eccezioni lanciate in dipendenza dei parametri attuali.
2. Si assuma di implementare la classe `Individual` con una struttura dati strutturata nel modo seguente

```
private Profile mySelf;
private ArrayList<Profile> myPublicContacts;
private ArrayList<Profile> myPrivateContacts;
// altri parametri per scelta pubblico/privato
public Individual(Profile my) { // parametri ignorati
   myself = my;
   myPublicContacts = new ArrayList<Profile>;
   myPrivateContacts = new ArrayList<Profile>;
}
```

   - Definire l'invariante di rappresentazione.
   - Implementare il metodo `share` verificando che preservi l'invariante di rappresentazione.

**Esercizio 3.** Si consideri il seguente programma con una sintassi tipo OCaml

```
let d = fun x -> x+x;;
d(5*25);;
```

1. Descrivere le differenze di valutazione della chiamata `d(5*25)` nel caso di **a)** passaggio dei parametri per valore, e **b)** passaggio dei parametri per nome.