PROGRAMMAZIONE II (A,B) - a.a. 2013-14
Terzo Appello — 5 Settembre 2014

Esercizio 1. Si consideri il seguente programma OCaml:

```
let x = 1;;
let f y = let x = y + 1 in fun z -> x+y+z;;
let x = 3;;
let g = f 4;;
let y = 5;;
let z = g 6;;
```

1. Indicare il tipo inferito dall'interprete OCaml per le funzioni `f` e `g`.
2. Indicare il valore restituito dal programma.
3. Simulando la valutazione del programma, mostrare la struttura della pila dei record di attivazione subito dopo l'invocazione di `let g = f 4`, e al momento dell'invocazione di `g 6`.
4. Assumendo di considerare una versione di OCaml con regole di scoping dinamico, mostrare la struttura della pila dei record di attivazione subito dopo l'invocazione di `let g = f 4`, e al momento dell'invocazione di `g 6`.

Esercizio 2. Una *Map* è una struttura dati il cui scopo principale è quello di rendere veloci ed efficienti operazioni quali inserimento e ricerca di elementi. Una *Map* può essere definita astrattamente come una collezione di coppie (chiave, valore) che permette di memorizzare e ricercare degli elementi in base al valore di una chiave.

Si consideri la seguente specifica parziale (come interfaccia Java) del tipo di dati astratto `Map`, generica rispetto ai parametri `<K,V>` che rappresentano rispettivamente il tipo delle chiavi e il tipo dei valori contenuti nella `Map`.

```
public interface Map<K,V> {
    // numero di elementi nella struttura
    public int size();
    // inserisce un nuovo elemento di chiave 'key' e valore 'value',
    // se la chiave non e' gia' usata
    public V add (K key, V value);
    // restituisce l'indice della coppia avente come chiave il valore 'key'
    // se esiste, altrimenti restituisce il valore -1
    public int indexOf(K key);
    //restituisce una lista con tutte le chiavi di this
    public List<K> getKeys();
    //rimuove l'elemento di chiave 'key', se esiste
    public boolean remove (K key);
}
```

1. Completare la specifica, descrivendo l'overview del tipo di dati astratto (compreso una rappresentazione di un elemento tipico) e indicando per ogni metodo: **a)** le clausole REQUIRES, MODIFY ed EFFECT; **b)** il valore restituito e le eventuali eccezioni lanciate in dipendenza dei parametri attuali.

Si assuma di implementare la specifica `Map` con la classe `MapImpl` che utilizza una coppia di liste per la rappresentazione delle chiavi e dei valori:

```
private List<K>  keys;
private List<V>  values;
```

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

3. Implementare i metodi `add` e `indexOf`, verificando che preservino l'invariante di rappresentazione

4. Supponiamo di implementare il metodo `getKeys()` nel modo seguente:

```
public List<K> getKeys() {
  return keys;
  }
```

   Si discuta se la collezione `Map` rispetta le caratteristiche essenziali della progettazione e realizzazione di astrazioni sui dati.

5. Supponiamo di estendere la collezione `Map` con il metodo `public V get(K key)` che restituisce il valore associato a una chiave. Consideriamo due differenti specifiche del metodo `get`

```
/** Specifica 1 **/
REQUIRES key != null e key e' una chiave in this
RETURN il valore V associato alla chiave K in this

/** Specifica 2 **/
REQUIRES key e' una chiave in this
RETURN il valore V associato alla chiave K in this
THROWS NullPointerException if key = null
```

   Discutere le due specifiche del metodo `get`.

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 *riferimento*. 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 riferimento e la loro invocazione.

2. Definire le regole OCaml dell'interprete per trattare la valutazione di dichiarazione di procedure con parametri per riferimento e la loro invocazione.