PROGRAMMAZIONE II (A,B) - a.a. 2013-14
Secondo Appello — 10 luglio 2014

Esercizio 1. Si consideri il seguente programma OCaml, che realizza l'elevazione a potenza (la funzione `power`) con moltiplicazioni successive:

```
let rec iterate n f d =
  if n = 0 then d
  else   iterate (n-1) f (f d)    (**)

let power i n =
  let i_times a = a * i in
    iterate n i_times  1

# power 3 2;;
- : int = 9
```

1. Indicare il tipo inferito dall'interprete OCaml per le funzioni `iterate` e `power`.
2. Quante volte viene eseguita l'istruzione marcata con `(**)` valutando l'espressione `power 3 2`?
3. Simulando la valutazione dell'espressione `power 3 2`, mostrare la struttura della pila dei record di attivazione subito dopo l'invocazione di `f` e subito dopo l'invocazione di `iterate` per ogni esecuzione della linea marcata con `(**)`.
4. La funzione `iterate` è *tail-recursive*. Descrivere informalmente come può essere sfruttata questa proprietà per ottimizzarne l'implementazione, e come questo modifichi le pile dei record di attivazione del punto precedente.

Esercizio 2. Un *Dizionario* è una struttura dati adatta a memorizzare e ricercare degli elementi di un tipo assegnato in base al valore di una chiave.

Si consideri la seguente specifica parziale (come interfaccia Java) del tipo di dati astratto `Dizionario`, generica rispetto ai parametri `<K,V>` che rappresentano rispettivamente il tipo delle chiavi e il tipo dei valori contenuti nel dizionario. L'interfaccia `Dizionario` utilizza una classe di tipo record `Elemento` (vedi sotto), anch'essa generica, in cui ogni oggetto ha una chiave e un valore.

```
public interface Dizionario<K,V> {
    // numero di elementi nel dizionario
    public int size();
    // inserisce un nuovo elemento di chiave 'key' e valore 'value',
    // se la chiave non e' gia' usata
    public boolean insert (K key, V value);
    // restituisce l'elemento di chiave 'key', se esiste
    public Elemento<K,V> find(K key);
    //restituisce una lista con tutti gli elementi di this
    public List<Elemento<K,V>> elementi();
    //rimuove l'elemento di chiave 'key', se esiste
    public boolean remove (K key);
}
```

```
public class Elemento <K,V>{
    public K key;
    public V value;

    public Elemento(K key, V value){
        this.key = key;
        this.value = value;
    }
}
```

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 `Dizionario` con la classe `DizionarioImpl` che utilizza una lista concatenata, sfruttando a seguente classe interna:

```
private class Node{
    Elemento<K,V>  elem;
    Node next;
    Node(Elemento<K,V> newElem, Node node){
        elem = newElem;
        next = node;
    }
}
```

2. La struttura di implementazione deve avere almeno una variablile `head` di tipo `Node` che punta alla testa dalla lista. Descrivere eventuali altre componenti della struttura di implementazione.

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

4. Implementare i metodi `insert`, `find` e `size`, verificando che preservino l'invariante di rappresentazione

5. Si consideri la specifica del TdA `DizionarioOrdinato` che estende quella di `Dizionario` modificando solo l'overview e il metodo `elementi()`. In particolare, in un `DizionarioOrdinato` sulle chiavi è definito un ordinamento totale, e il metodo `elementi()` restituisce tutti gli elementi in ordine crescente di chiave.[1] Si discuta se il TdA `DizionarioOrdinato` rispetta il Principio di Sostituzione rispetto al TdA `Dizionario`.

Esercizio 3. Si consideri il linguaggio didattico funzionale. Estendiamo il linguaggio in modo da includere espressioni e valori di tipo **record**. Un *valore (di tipo) record* è un dato strutturato composto da un numero finito di coppie *nome-valore*, detti **campi**. Analogamente, un'*espressione (di tipo) record* è composta da un numero finito di coppie *nome-valore*. La valutazione di un'espressione record produce un valore record.

Un identificatore può essere legato a un valore record tramite il costrutto `let`, come nel seguente esempio dove (in sintassi OCaml-like) si evidenzia che l'espressione record che compare nel `let` viene valutata in un valore record:

```
let rect  = record{base = 5 * 5, altezza = 10 - 6}

> val rect = record{base = 25, altezza = 4}
```

Sui record èdefinita l'operazione di selezione di una componente. Per esempio, continuando l'esempio precedente:

```
let b  = rect.base
> val b = 25
```

1. Estendere la sintassi astratta del linguaggio didattico funzionale in modo da includere valori ed espressioni record e l'operatore di selezione.

2. Estendere il codice OCaml dell'interprete per trattare la valutazione di espressioni record e dell'operatore di selezione.

[1] Naturalmente l'ordinamento sulle chiavi può essere sfruttato per implementare in modo più efficiente alcuni metodi dell'interfaccia, ma questo non è di interesse per questo esercizio.