PROGRAMMAZIONE II (A,B) - a.a. 2019-20
Seconda Valutazione Intermedia – 19 Dicembre 2019 Soluzione Parziale

Domande di base

- Si consideri il seguente programma Java

```
class A {
    public void foo (Object o) { System.out.println("A"); }
}
class B {
public void foo (String o) { System.out.println("B"); }
}

class C extends A {
public void foo (String s) { System.out.println("C"); }
}
class D extends B {
public void foo (Object o) { System.out.println("D"); }
}

class Main {
public static void main(String[] args) {
   A a = new C(); a.foo("Java");
   C c = new C(); c.foo("Java");
   D d = new D(); d.foo("Java");
   B b = new D(); b.foo("Java");
   }
}
```

  1. Si descriva la struttura della tabella dei metodi (*vtable*) per tutte le classi presenti nel programma. Motivare la risposta.
  2. Si descriva l'ordine in cui sono caricate le classi durante l'esecuzione del metodo `main`. Motivare la risposta.

     In Java una classe viene caricata e inizializzata quando un suo oggetto (o un oggetto che appartiene a una sua sotto-classe) è referenziato per la prima volta. Quindi l'ordine di caricamento del metodo ***`main`*** *è* `A, C, B,D`

- Spiegare il ruolo del meccanismo della *retention* nell'implementazione dei linguaggi funzionali. *Il meccanismo della retention si usa nei linguaggi funzionali con regola di scope statico per implementare le funzioni di ordini superiore che restituiscono una funzione come risultato. In questo caso l'ambiente non locale potrebbe contenere riferimenti a valori contenuti nel record di attivazione della funzione che restituisce il risultato ed anche nel suo ambiente locale (raggiungibile con il puntatore di catena statica). Di conseguenza la chiusura che modella la funzione come risultato contiene dei riferimenti a record di attivazione presenti sulla stack che andrebbero rimossi dopo avere restituito il risultato seguendo la normale politica FIFO.*

- Descrivere l'impatto delle regole di *scope* nell'implementazione dei linguaggi di programmazione.

- Tutte le variabili locali del metodo `main` di una applicazione Java appartengono al *rootset*? Si motivi la risposta.

  Il root set contiene le variabili statiche e le variabili allocate sul runtime stack e quindi anche tutte le variabili locali del metodo ***`main`****. Questa informazione viene usata dal garbage collector per determinare i dati ancora attivi ovvero quelli raggiungibili anche indirettamente dal root set seguendo i puntatori.*

Esercizio 1

Si consideri il seguente programma OCAML

```
let length list =
    let rec aux n = function
      | [] -> n
      | _::t -> aux (n+1) t
    in aux 0 list;;
let apply = fun a -> ( fun b -> a b );;
let k = 100;;
let f = fun x -> x+k;;
let g = fun x -> x-k;;
let k = 150;;
let l = [10;15;20];;
apply (if (length l  > 2) then g else f ) k;;
```

1. Si simuli la valutazione del programma mostrando la struttura della pila dei record di attivazione.
2. Assumendo di adottare una regola di scoping dinamico, si descriva quali sono le modifiche che avvengono nella struttura a run-time dei record di attivazione durante l'esecuzione del programma. *Con scope statico il programma restituisce come valore 50 mentre adottando una regola dinamica restituisce 0. Infatti la funzione* `apply` *applica la funzione* `g` *al valore di k nell'ambiente della chiamata che e' usato anche nel corpo di g per valutare l'espressione* $x - k$.

Esercizio 2

Estendere il linguaggio didattico funzionale con il tipo di dato `IntSet` per poter operare con insiemi finiti di valori interi mediante un insieme di operatori primitivi quale `create, isEmpty, insert, remove, isIn` con il significato ovvio.

1. Definire le regole di valutazione delle operazioni primitive su insiemi di valori interi e estendere consistentemente la struttura dell'interprete del linguaggio didattico funzionale.

   Si presenta una soluzione in cui `insert` *aggiunge sempre un intero alla lista, mentre* `remove` *ne rimuove tutte le occorrenze.*

```
type exp = ...  | IntSet of set | Create of set |  isIn of exp * exp |

      isEmpty of exp  |  insert of exp * exp | remove of exp * exp
   ...
and set = Empty | Item of exp * set

type evT = ...| IntSetVal of int list

let rec eval (e : exp) (r : evT env) : evT = match e with
   ...
   | IntSet (s) -> IntSetVal (evalSet s r)
   | Create (s) ->  IntSetVal (evalSet s r)
   | isEmpty(s) ->   match eval s r with
                  IntSetVal (l) ->  match l  with
                                      []  -> true
                                    | h:: tl ->  false
                  | _ -> failwith("non insieme")
   | isIn(s,e1) ->   match eval s r  with
                  IntSetVal (l) ->  match eval e1 r with
                                       Int i ->  lookup l i
                                       | _ -> failwith("non intero")
                  | _ -> failwith("non insieme")

   | insert (s, a)  =  match (eval s r, eval a r) with
                    ( IntSetVal l, Int i ) -> IntSetVal (i :: l)
                      | _, _ -> failwith("errore") )
   | remove(s, a) = match (eval s  r, eval a r) with
              (IntSetVal l, Int i ) -> IntSetVal (removeSet l i)
              | _, _ -> failwith("errore") )
         ...
and let rec evalSet (s : set) (r : evT env) : int list = match s with
     Empty -> []
   | Item (e1, s1) ->  match eval e1 r with Int i  -> i ::(evalSet s1 r)
                                  | _ -> failwith("non intero ")

and let rec removeSet (l : int list) (i: int) : int list = match l with
     [] -> []
   | h :: tl  -> if (i = h ) then (removeSet tl i ) else h  :: (removeSet tl i )

and let rec lookup   l i  = match l with
        [] -> false
       | h:: tl  -> if i=h then true else lookup tl i
```

2. Introdurre una nuova tipologia di astrazione funzionale che consente di applicare una funzione a tutti gli elementi di un insieme di valori interi. Assumiamo per semplicità che l'astrazione introdotta non sia ricorsiva. Supponiamo inoltre, che la sintassi concreta dell'astrazione funzionale abbia la forma `FunOverSet(x:int, s:  set) = body`. Mostrare come deve essere modificato l'interprete del linguaggio didattico per includere questa nuova forma di astrazione funzionale.

```
type exp = ... Fun of ide * exp | Apply  of exp * exp |
                                  IntSet of set | FunSet of ide * ide * exp
      ----
and set = Empty | Item of exp * set

type evT = ...FunVal of evFun |
                    IntSetVal of int list |
                    FunSetVal of ide * ide * exp * evT env

and evFun = ide * exp * evT env

let rec eval (e : exp) (r : evT env) : evT = match e with
   ...
   | IntSet (s) -> IntSetVal (evalSet s r)
   ....
   | FunSet (i, s, a)   -> FunSetVal(i, s, a, r)
   | Apply(f ,a)  ->
         let fClosure = (eval f r) in
            match fClosure with
                                 ...................
             FunSetVal(x, s, body, r1) ->
                         match eval a r with
                           IntSetVal(l) ->
                            let rec  applyFun  l   =
                                    match l with
                                          [] -> []
                                       | h :: tl  ->
                                         let par = eval h r   in
                                          eval body  (bind r1 x par ) :: applyFun tl
                                             in IntSetVal  (applyFun l )
                            _ -> failwith("non insieme")) |

        _ -> failwith("non functional value"))
```