

PROGRAMMAZIONE II (A,B) - a.a. 2018-19
Seconda Valutazione Intermedia – 19 Dicembre 2018
Soluzione

Domande di Base

1. Descrivere la struttura e le informazioni presenti nel record di attivazione dell'implementazione di un linguaggio con le caratteristiche di OCaml. *Soluzione in classe*
2. Descrivere la tecnica di gestione della memoria Heap denominata *contatori di riferimento*. *Soluzione in classe*
3. Spiegare il motivo per cui la regola di *scoping dinamico* comporta la necessità di avere un meccanismo di type checking dinamico. *Soluzione in classe*

Esercizio 1

Si consideri il paradigma funzionale. Introduciamo un nuovo costrutto linguistico che permette di comporre e applicare una sequenza di funzioni. In particolare introduciamo il costrutto `pipe` che permette di comporre una sequenza di n funzioni, invocandole sequenzialmente, da sinistra a destra, e chiamando ogni funzione con il risultato dell'invocazione precedente. Per esempio, si consideri il seguente frammento di codice

```
let plusOne = fun y -> 1 + y;;
let plusTwo = fun y -> 2 + y;;
pipe(plusOne, plusTwo, 6);;
```

Parte a

L'invocazione di `pipe(plusOne, plusTwo, 6)` produce come risultato il valore 9. Vengono applicate in sequenza le due funzioni `plusOne`, `plusTwo` a partire dall'argomento attuale 6.

1. Si estenda la sintassi astratta del linguaggio didattico funzionale in modo da includere il costrutto `pipe(Lista-funzioni, argomento)`.

Assumendo di modellare solo funzioni unarie, come nell'interprete presentato in classe

```
type exp = ...
  | type exp = ...
  | Pipe of funArgs * exp
  ...
and funArgs = Empty | FunPar of exp * funArgs
```

2. Si discutano i vincoli di tipo del costrutto `pipe(Lista-funzioni, argomento)` che evitano la generazione di errori a tempo di esecuzione. Supponiamo di avere `Pipe (flist ,e)` la prima funzione della lista (se esiste) deve prendere in input un valore del tipo di `e` e produrre un valore del tipo preso come input seconda della seconda funzione (se esiste). Questo deve valere ricorsivamente per tutte le funzioni della lista.
3. Si definiscano le regole OCaml dell'interprete per trattare la valutazione di `pipe(Lista-funzioni, argomento)`.

```

let rec eval (e : exp) (r : evT env) : evT = match e with
  ...
  | Pipe (fArgs e) -> let aFunArgs = evalFA fArgs r in
    let av = eval e r in
    applyAll aFunArgs av
  ...
and let rec evalFA (f : funArgs) (r : evT env) : evT list = match f with
  Empty -> failwith("funagrs error")
  | FunPar (e, fs) -> let af = (eval e r) in
    if type(af, "fun") then af::(eval fs r) else failwith("funagrs error")
and let rec applyAll (f : evT list) (v: evT) : evT = match f with
  FunVal(i, body ,decenv) :: [] -> eval body (bind i v decenv)
  | FunVal(i, body, decenv):: fs ->
    let acc = eval body (bind i v decenv) in applyAll fs acc

```

Parte b

Consideriamo ora un linguaggio funzionale OCaML-like esteso con il costrutto linguistico pipe descritto in precedenza.

```

let n = 4;;
let strange = fun l ->
  match l with
  [ ] -> fun x -> n + x
  | [hd] -> fun x -> n + x + hd
  | hd::(hd1::ls) -> fun x -> n + x + hd + hd1;;
let rec sum l =
  match l with
  | [] -> 0
  | hd :: tl -> hd + sum tl
let alist = [1;2;0;3];;
pipe(sum, (strange alist), alist)

```

1. Tenendo conto della regola di valutazione definita nel punto precedente, si simuli la valutazione del programma mostrando la struttura della pila dei record di attivazione: *in classe*
2. Si determini il valore calcolato dal programma: *13*

Esercizio 3

Si consideri il seguente programma scritto in una notazione Java-like.

```

public class B {
  public void foo(B obj) { System.out.print("B1");
  }
  public void foo(C obj) { System.out.print("B2");
  }
}
class C extends B {
  public void foo(B obj) { System.out.print("C1");
  }
  public void foo(C obj) { System.out.print("C2");
  }
}

```

1. Si descriva, motivando la risposta, la struttura delle tabelle dei metodi (Dispatch vector) delle due classi supponendo di adottare la tecnica denominata *sharing strutturale*. Con lo *sharing strutturale* il Dispatch vector di ogni classe contiene tutti i metodi ereditati e quelli introdotti nella classe. Quindi avremo

Class B	<i>Metodi</i>
foo(B obj)	System.out.print("B1")
foo(C obj)	System.out.print("B2")

Class C	<i>Metodi</i>
foo(B obj)	System.out.print("C1")
foo(C obj)	System.out.print("C2")