PROGRAMMAZIONE II
Esercitazione

**Es1.** Si aggiunga al linguaggio didattico, frammento imperativo, il costrutto iterativo **do** E1 ; C1 — E2 ; C2 **od** dove E1 e E2 sono espressioni di tipo bool. Il comando ha il seguente comportamento:

- si valutano le espressioni,
- se nessuna della due espressioni viene valutata a true si esce dall'iterazione;
- se una espressione viene valutata a true si esegue esegue il comando relativo e poi si continua l'iterazione;
- se entrambe l'espressioni vengono valutate a true, viene selezionato in modo casuale il comando da eseguire, si esegue e poi si continua l'iterazione.

Si mostri come deve essere modificata la funzione di valutazione semantica dei comandi e l'interprete iterativo del frammento imperativo.

**Es2.** Si estenda il frammento funzionale del linguaggio didattico con un costrutto per la dichiarazione parallela di variabili: `letpar I1=E1 and I2=E2 in E`.
L'espressione `letpar(i1, e1, i2, e2, e)` denota la sintassi astratta.
La dichiarazione parallela ha il seguente comportamento: si valutano le espressioni E1 e E2 nello stesso ambiente e poi si esegue E nell'ambiente esteso con i legami tra gli identificatori I1, I2 e il risultato della valutazione di E1, E2.

- Si mostri come deve essere modificata la funzione di valutazione semantica delle espressioni.
- Si mostri come deve essere modificato l'interprete iterativo del frammento funzionale.
- Dire sotto quali condizioni l'esecuzione dell'espressione `letpar I1=E1 and I2=E2 in E` e' equivalente all'esecuzione dell'espressione `let I1=E1 in let I2=E2 in E`. Si motivi la risposta,

**Es3.** Tradurre ogni uso degli identificatori con la coppia (numero di passi da effettuare sulla catena statica, posizione relativa) dei seguenti programmi funzionali (con scoping statico).

```
let f x = x - 1 in
          let f x = f (x - 1) in
               f 2

let f x = x - 1 in
     let rec f x = if x = 0 then 1 else f (x - 1) in
               f 2
```

**Es1 Traccia soluzione** La funzione di valutazione semantica dei programmi puo' essere modificata nel modo seguente:

```
do(e0, c0, e1, c1) -->
   let g0 = sem(e0, r, s) in
        let g1 = sem(e1, r, s) in
             if typecheck(bool g0) && typecheck(boll g2) then
                 (match (g0, g1) with
                  | (Bool(true), Bool(false)) -> semcl(c0 @ do(e0,c0, e1, c1))
                  | (Bool(false), Bool(true)) -> semcl(c1 @ do(e0,c0, e1, c1))
                  | (Bool(true), Bool(true)) -> pickup(c1, c2, do(e0,c0, e1, c1))
                  | (Bool(false), Bool(false)) -> s)
              else failwith ("nonboolean guard")


let pickup(c0, c1, do(e0,c0, e1, c1)) =
     let n = Random.int 2 in
          match n with
          | 0 -> semcl(c0 @ do(e0,c0, e1, c1))
          | 1 -> semcl(c1 @ do(e0,c0, e1, c1))
```

Per quanto riguarda l'interprete iterativo. Una possibile soluzione risulta:

```
(* interprete iterativo *)

| Com1(do(e0, c0, e1, c1)) -> pop(continuation);
                                                    push(Coml(Com2 do(e0, c0, e1, c1))::newc
                                  push(Expr1(doexp(e0,e1))), continuation)

| Com2(do(e0, c0, e1, c1)) ->
        let (g0, g1) = top(tempstack)  in
             pop(tempstack);
              (match (g0, g1) with
                   | (Bool(true), Bool(false)) ->
                       (let old = newcl in let newl =
                  (match labelcom c0 with
           | Com1 newl1 -> newl1
        | _ -> failwith("impossible")) in
                                let nuovo =
                                                  Coml(newl @ [Com1(do(e0,c0, e1, c1)] @ old)
                                                            pop(continuation); push(nuovo,cont
(* simile agli altri casi*)
                   | (Bool(false), Bool(false)) -> ()
```

```
| _  failwith ("nonboolean guard"))
```