Corso di Paradigmi di Programmazione
Prova scritta del 1 febbraio 2012.

Tempo a disposizione: ore 2.

1. Si dice in genere che Java è un linguaggio interpretato. Si spieghi sinteticamente, ma con precisione, cosa si intende dire con tale espressione.

2. Con la notazione $\mathcal{C}^L_{L_1,L_2}$ indichiamo un compilatore da L_1 a L_2 scritto in L. Con $\mathcal{I}^L_{L_1}$ indichiamo un interprete scritto in L per il linguaggio L_1. Si dica se la seguente scrittura ha senso
   $\mathcal{I}^L_{L_1}(\mathcal{I}^{L_1}_L, (\mathcal{C}^L_{L,L_1}(P^L)))$.
   Se la risposta è "no" si motivi tale fatto; se è "sí", si dica qual è il risultato ottenuto.

3. Data una grammatica $G = (NT, T, S, P)$ si dia con precisione la definizione di "*linguaggio generato da G*".

4. Si consideri un linguaggio con scope statico per tutti i nomi, nel quale i nomi delle eccezioni devono essere dichiarati con la sintassi

   `exception` *`nomeeccezione`*`;`

   Qual è l'effetto dell'esecuzione del seguente frammento?

```
exception X;

void f(){
    throw X;
 }

void B(){
    exception X;
    void g(){
        try {f();} catch (X){
                      print("in␣g");
                      }
    }
}

B();
```

5. Si dica cosa viene stampato dal seguente frammento di codice scritto in uno pseudo-linguaggio che usi scoping dinamico e deep binding:

```
int x = 3;
procedure stampa_x(){
    write_integer(x);
    }
procedure ass_x(n:int){
    x = n;
    if (n=1) stampa_x();
    }
procedure pippo(function S, P ; int n){
    int x = 10;
    if (n=1) then  ass_x(n)
             else {S(n);
                   P();
             }
    }

pippo(ass_x, stampa_x, 1);
pippo(ass_x, stampa_x, 2);
```

6. Usando uno pseudolinguaggio che usi i puntatori si fornisca un frammento di codice che generi un "dangling reference". Si faccia quindi vedere come con la tecnica delle "tombstone" non si ha più tale problema.

7. Si consideri l'implementazione dello scope statico mediante display. Si dica, motivando la risposta, se dimensione massima del display può essere determinata durante la compilazione.

8. Si supponga che `B_exp` e `Stat` siano non terminali di una grammatica libera già definiti e che `if, then, endif, else` siano terminali. Supponiamo che il non terminale IfThenE indichi il comando

alternativo e che un linguaggio di programmazione A che abbia la seguente definizione per tale comando:

IfThenE ::= **if** `B_exp` **then** `Stat` **endif** `|`

if `B_exp` **then** `Stat` **else** `Stat` **endif**

mentre un linguaggio di programmazione B abbia la seguente definizione

IfThenE ::= **if** `B_exp` **then** `Stat` `|`

if `B_exp` **then** `Stat` **else** `Stat`

Si dica, motivando le risposte se: 1) le due regole definiscono la stessa sintassi (ossia se le corrispondenti grammatiche generano lo stesso linguaggio formale); 2) supponendo che ci sia anche la regola

Stat:: = IfThenE

se ci sono dei vantaggi nella scelta di A o di B.