Prova Scritta di Linguaggi di Programmazione I

13/01/2006

1. Sia $C_{L_1}^{L_2 \to L_3}$ un compilatore da L_2 a L_3 scritto in L_1 e $I_{L_1}^{L_2}$ un interprete di L_2 scritto in L_1. Sia $[\![P]\!]$ la funzione calcolata dal programma P. Si dica cosa produce la valutazione $[\![I_{L_1}^{L_2}]\!](C_{L_2}^{L_3 \to L_2}, C_{L_3}^{L_4 \to L_3})$ e se tale risultato può effettivamente esistere.

2. Si stabilisca se la grammatica

$$
\begin{array}{ll}
S ::= A \mid AS \mid BR \mid CT & A ::= 0 \mid 3 \mid 6 \mid 9 \\
R ::= AR \mid BT \mid C \mid CS & B ::= 1 \mid 4 \mid 7 \\
T ::= AT \mid B \mid BS \mid CR & C ::= 2 \mid 5 \mid 8
\end{array}
$$

   è ambigua. Si caratterizzi inoltre il linguaggio generato.

3. Si mostri l'output del seguente frammento di programma espresso in un linguaggio C-like con assegnamento che calcola l'r-value *prima* dell'l-value e indici dei vettori che iniziano da 0:

```
{
    char x[10] = il_proprio_cognome;
    int i = 1;
    char magic(ref char y, ref int j)
    {
        char c = y;
        x[j] = x[j--] = x[j--];
        write(y);
        return c;
    }
    write(magic(x[i++], i));
    write(x[i],i);
}
```

4. Si mostri l'output del seguente frammento di programma espresso in un linguaggio C-like con scoping *statico* e valutazione delle espressioni da sinistra a destra:

```
{
    int x = 1;
    int y = 0;
    proc P(valueresult int y, ref int z, int R(name int))
    {
        z = y-- + R(x++ + y++);
        write(x,y,z);
        z = R(z++);
    }
    {
        int x = 3;
        int Q(name int w)
        {
            return (w + x++ + y++);
        }
        P(x, y, Q);
        write(y++,x++);
    }
    write(y,x);
}
```

$2, 3, 9, 23, 3, 24, 2$

5. Con riferimento alla tecnica di implementazione con catena statica e "display", per l'esercizio precedente si mostri (schematicamente) con dei diagrammi la situazione sul display e sullo stack di sistema al momento in cui viene eseguita Q.

Prova Scritta di Linguaggi di Programmazione I

13/01/2006

6. Data la seguente definizione del tipo di dato astratto (polimorfo) *Matrici* **type** Matrix a = [[a]] si scriva un predicato Haskell uppertriangular :: Matrix a −> **Bool** che determina se una matrice (quadrata) è triangolare superiore. Si implementino le matrici per righe come liste di liste.

   A titolo di esempio,
   `uppertriangular([[1,0,-1],[0,2,3],[0,0,1]])` riscrive a `True`, mentre
   `uppertriangular([[1,-1,0],[0,2,3],[5,0,1]])` riscrive a `False`.

   Si scriva il programma con variabili anonime ove possibile.

7. Data la seguente definizione del tipo di dato astratto (polimorfo) *Binary Search Tree*

```
data (Ord a) => BST a =
    Void | Node a (BST a) (BST a)
    deriving (Eq, Ord, Read, Show)
```

   si scriva un predicato Haskell `isBST` che dato un albero verifica se i valori in esso contenuti soddisfano la proprietà strutturale dei Binary Search Trees.

   Si dia espressamente il tipo di `isBST`.