11 settembre 2006

# Linguaggi di Programmazione

| | |
|---|---|
| *Nome e Cognome* | |
| *Corso di laurea* | |
| *Telefono* | |
| *Email* | |

1. Specificare la BNF di un linguaggio imperativo in cui ogni programma, identificato da un nome, contiene zero o più dichiarazioni di variabili ed un corpo, come nel seguente esempio:

```
program esempio
  int a, alfa, beta4;
  float x, y, z;
  string s1, s2;
  int i;
begin
  alfa := 10;
  beta4 := alfa;
  if alfa > 3 then
    x := 12.1;
    y := 1.98;
  else
    z := x;
  endif;
  s1 := "alfa";
  repeat
    a := alfa;
    x := a;
  until x = alfa;
end.
```

Le variabili possono essere di tipo **`int`**, **`float`** o **`string`**. Il corpo del programma è costituito da una sequenza non vuota di istruzioni racchiusa tra **`begin`** ed **`end`**. Ogni istruzione può essere un assegnamento, una istruzione condizionale (a una o due vie) o un ciclo a condizione finale. Possono essere assegnate variabili con altri variabili o valori. Una condizione è il confronto (**`=`**, **`!=`**, **`>`**, **`<`**) tra una variabile e un valore o un'altra variabile.

2. È data la seguente tabella di operatori per la quale si assume priorità decrescente dall'alto verso il basso:

| *Operatore* | *Tipo* | *Associatività* | *Ordine valutazione* | *Corto circuito* |
|---|---|---|---|---|
| **`not`** | unario | destra | - | - |
| **`and`** | binario | sinistra | da destra a sinistra | no |
| **`or`** | binario | destra | da sinistra a destra | si |

e la seguente espressione:

```
a or b and c or d or not e
```

a) Rappresentare l'albero della espressione.
b) Specificare la semantica operazionale della espressione.

NB: Il linguaggio di specifica operazionale è così caratterizzato:

- Contiene gli operatori di negazione (**`!`**), congiunzione (**`&&`**) ed assegnamento (**`:=`**).
- Non contiene l'operatore di disgiunzione, né le costanti logiche **`true`**, **`false`**;
- Ogni operatore non può essere applicato ad espressioni, ma solo a variabili;
- Contiene le istruzioni condizionali (*if-then* ed *if-then-else*) i cui predicati possono essere solo variabili;
- Contiene l'istruzione **`return`** il cui argomento è la variabile che contiene il valore della espressione;
- L'esecuzione della **`return`** termina immediatamente l'esecuzione del programma di specifica operazionale.

3. Definire nel linguaggio *Scheme* la funzione `remove`, avente in ingresso una funzione `f` ed una `lista`. Si assume che la funzione `f` sia unaria e computi un valore booleano. La funzione `remove` restituisce la sotto-lista (eventualmente vuota) di `lista` comprendente gli elementi che rendono la funzione `f` falsa, come nei seguenti esempi:

| **`lista`** | **`f`** | **`(remove f lista)`** |
|---|---|---|
| `()` | `number?` | `()` |
| `(x 2 4 (y z))` | `number?` | `(x (y z))` |
| `(x 10 () y (1 x 20) 82)` | `atom?` | `(() (1 x 20))` |
| `(a (b c) 10 (1 2 x))` | `list?` | `(a 10)` |
| `(a (b (1 2 y) 3) 25 () z w)` | `list?` | `(a 25 z w)` |
| `(a b 20)` | `list?` | `(a b 20)` |

4. Definire nel linguaggio *Haskell* la funzione `exists`, che riceve una `lista` di elementi ed una funzione booleana `f` applicata al tipo degli elementi di `lista`. La funzione `exists` restituisce `True` se `lista` contiene almeno un elemento che rende vera la `f`, altrimenti restituisce `False`.

5. Definire nel linguaggio *Prolog* il predicato `semisomma(L,S)`, che risulta vero quando `S` rappresenta la sommatoria degli elementi della lista `L` in posizione dispari. Per definizione, la semisomma di una lista vuota è zero.

6. Illustrare le differenze sostanziali tra un linguaggio funzionale (come Haskell) ed un linguaggio logico (come Prolog).