## Linguaggi di Programmazione (III Anno), Giugno 1998 - OOP

Si puo' usare una sintassi sia C++ che Java-like (ma si consiglia l'uso di Java). Per le parti di codice che sono indipendenti dagli aspetti object-oriented di tali linguaggi si consiglia di usare pseudo-istruzioni o una descrizione a parole.

**Esercizio OOP.1 (7 punti)** Si implementi la classe *persona* e *persone* (che corrisponde ad una lista di persone).

```
class persona {
...
public persona(persona padre, persona madre) DA IMPLEMENTARE (COSTRUTTORE)
public persone figli()                       DA IMPLEMENTARE
}
class persone {
...
public persone(persona primo, persone altri) DA IMPLEMENTARE (COSTRUTTORE)
public persona primo()                       DA IMPLEMENTARE
public persone altri()                       DA IMPLEMENTARE
}
```

Al momento della nascita di persona (creazione di un oggetto persona) si dichiara di chi e' padre e madre, quindi tale persona va aggiunta all'insieme dei figli di padre e madre.

**Esercizio OOP.1 (8 punti)** Si implementino le seguenti interfacce con opportune classi *Segmento*, *Rettangolo* e *Parallelepipedo*. Tali classi devono avere dei costruttori che prendono rispettivamente 1, 2 e 3 parametri di tipo float. Si cerchi di riutilizzare il codice mediante inheritance, implementando *Parallelepipedo* come sottoclasse di *Rettangolo*, e *Rettangolo* come sottoclasse di *Segmento*.

```
interface SegmentoInt {
float lughezza();
}
interface RettangoloInt {
float lughezza();
float area();
}
interface ParallelepipedoInt {
float lughezza();
float area();
float volume();
}
```

## Linguaggi di Programmazione (III Anno), Giugno 1998 - FP

**Esercizio FP.1 (4 punti)** Dire se le seguenti espressioni sono tipabili in ML (usando l'algoritmo W di Hindley-Milner), e in caso affermativo dare il tipo piu' generale.

1. $\lambda y, x.y\ (y\ x)$
2. $\lambda f.cons(nil, f\ nil)$

dove $cons : \forall X.X \to X\ list \to X\ list$ e $nil : \forall X.X\ list$.

**Esercizio FP.2 (7 punti)** Definire le operazioni $insert, delete : X \to X\ list \to X\ list$ e $union : X\ list \to X\ list \to X\ list$ per insiemi rappresentati con liste senza ripetizioni, dove $X$ varia sugli equality types (tali variabili in ML sono indicate con $''X$).
Si definisca un datatype $X\ Term$ corrispondente all'insieme dei $\lambda$-termini con variabili in $X$, cioe' $M ::= x \mid M_1 M_2 \mid \lambda x.M$. Quindi si definisca la funzione $FV : X\ Term \to X\ list$, t.c. $FV(M)$ e' la lista senza riperizioni delle variabili libere di $M$. (per definire $FV$ si usino le funzioni insiemistiche precedentemente definite).

```
fun insert ...
fun delete ...
fun union  ...
datatype ''X Term =...
fun FV ...
```

Si possono usare funzioni ausiliarie sulle liste (ma darne il tipo e descrivere a parole cosa fanno).

**Esercizio FP.3 (4 punti)** Si consideri il tipo di dato albero binario (con etichette in $X$). si definisca la funzione $paths : X\ Btree \to X\ list\ list$ che restituisce la lista di tutti i cammini dalla radice ad una foglia (cioe' $empty$). Dove un cammino e' dato dalla lista delle etichette dei nodi che lo compongono.

```
datatype 'X Btree = leaf of 'X | node of 'X * 'X Btree * 'X Btree;
fun paths ...
```