Linguaggi di Programmazione (III Anno), Aprile 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 consideri la seguente classe astratta *Set*, i cui oggetti rappresentano insiemi di interi. Quindi si implementino le due sottoclassi *Interval* e *Union*.

```
abstract class Set {
public abstract bool member(int x);
}
class Interval extends Set {
...
public Interval(int x, int y) DA IMPLEMENTARE
public bool member(int x)     DA IMPLEMENTARE
}
class Union extends Set {
...
public Union(Set X, Set Y) DA IMPLEMENTARE
public bool member(int x)  DA IMPLEMENTARE
}
```

Esercizio OOP.2 (8 punti) Si consideri la classe *Conto* che rappresenta un normale conto corrente. Si implementi la sottoclasse *Conto*0 che addebita gli interessi su un'altro conto, e la sottoclasse *Conto*1 che impone una tassa di 1 per ogni operazione di deposito e prelievo.

```
class Conto {
private float v;
public Conto() {v=0;}
public float controlla() {return v;}
public deposita(float x) {v=v+x;}
public preleva(float x) {v=v-x;}
public interessi() {v=1.1*v;} /* da il 10% di interessi */
}
class Conto0 extends Conto {
...
public Conto0(Conto X) DA IMPLEMENTARE
public interessi()     DA IMPLEMENTARE
}
class Conto1 extends Conto {
...
public Conto1 ()         DA IMPLEMENTARE
public deposita(float x) DA IMPLEMENTARE
public preleva(float x)  DA IMPLEMENTARE
}
```

Linguaggi di Programmazione (III Anno), Aprile 1998 - FP

Esercizio FP.1 (5 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. $\lambda x, y.y$
2. $\lambda x.cons(nil, x)$
3. $\lambda x, y.cons(yx, x)$

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

Esercizio FP.2 (5 punti) Si consideri il tipo di dato albero, quindi si definisca la funzione $TreeCompose$ che restituisce la composizione $f_1 \circ \ldots \circ f_n$ delle funzioni che si trovano sulle foglie dell'albero (f_1 e' la funzione piu' a sinistra e f_n quella piu' a destra), quando $n = 0$ tale composizione e' la funzione identita'.

```
datatype 'X tree = leaf of 'X | node of ('X tree) list;
val TreeCompose : ('X -> 'X) tree -> 'X -> 'X;
```

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

Esercizio FP.3 (5 punti) Si vuole astrarre dalla definizione di "tree" e "map" mediante un funtore $Tree$, in modo che applicandolo alla struttura $List$ si ottenga "tree" e la corrispondente funzione map. Completare il corpo della definizione del funtore.

```
signature SigC =
sig
    eqtype 'X C;
    val map : ('X -> 'Y) -> 'X C -> 'Y C;
end;
structure List : SigC =
struct
    type 'X C = 'X list;
    val map = "la usuale map per le liste"
end;
functor Tree (S:SigC) : SigC =
struct
    datatype 'X C = leaf of 'X | node of ('X C) S.C;
    fun map ...  <<============================== DA IMPLEMENTARE
end;
```