Linguaggi di Programmazione (III Anno), Aprile 1998 - OOP in Java

Esercizio OOP.1

```
abstract class Set {
public abstract bool member(int x);
}
class Interval extends Set {
private int u, v;
public Interval(int x, int y) {u=x; v=y;}
public bool member(int x) {return ((u<=x) and (x<=v));}
}
class Union extends Set {
private Set U, V;
public Union(Set X, Set Y) {U=X; V=Y;}
public bool member(int x)  {return (U->member(x) or V->member(x));}
}
```

Esercizio OOP.2

```
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 {
private Conto C;
public Conto0(Conto X) {C=X;}
public interessi()     {C->deposita(0.1*controlla());}
}
class Conto1 extends Conto {
public Conto1 () {INIZIALIZZAZIONE DELLA SUPERCLASSE, gia' fatto per default}
public deposita(float x) {super->deposita(x-1);}
public preleva(float x)  {super->preleva(x+1);}
}
```

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

Esercizio FP.1

1. $\lambda x, y.y$ ha tipo piu' generale $X \to Y \to Y$
2. $\lambda x.cons(nil, x)$ ha tipo piu' generale $(X\ list\ list) \to (X\ list\ list)$
3. $\lambda x, y.cons(yx, x)$ ha tipo piu' generale $(X\ list) \to ((X\ list) \to X) \to (X\ list)$

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

Esercizio FP.2

```
datatype 'X tree = leaf of 'X | node of ('X tree) list;
fun ListCompose nil x = x
  | ListCompose (f::l) x = f(ListCompose l x);
fun TreeCompose (leaf f) = f
  | TreeCompose (node l) = ListCompose (map TreeCompose l);
```

dove $map : (X \to Y) \to X\ list \to Y\ list$ e $ListCompose : ((X \to X)\ list)\ X \to X$

Esercizio FP.3

```
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 f (leaf x) = leaf (f x)
      | map f (node l) = node (S.map (map f) l);
end;
```