Linguaggi di Programmazione (III Anno), Febbraio 1998 - OOP in C++

Esercizio OOP.1

```
class Id {
static int n=0;
int i;
public:
Id() {i=n; n=n+1;}
Id(Id *x) {i=x->i; }
print() { stampa x_i; }
bool eq(Id *x) {return(i == x->i);}
}
```

Esercizio OOP.2

```
class Term {
public:
virtual print();
virtual Term sub(Sub S);
}

class Var: public Term {
Id y;
public:
Var (Id x) {y=x;}
print() {y->print();}
Term sub(Sub S) {return(S->fetch(y));}
}

class App: public Term {
Term s1,s2;
public
App (Term t1, Term t2) {s1=t1; s2=t2;}
print() {s1->print(); s2->print();}
Term sub(Sub S) {return(new App(s1->sub(S),s2->sub(S)));}
}

class Sub {
(Id*Term) list l;
public:
Sub() {l=nil;}
Term fetch(Id x) {if (x,t) in l then return(t) else return(new Var(x));}
update(Id x,Term t) {l=(x,t)::l;}
}
```

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

Esercizio OOP.1

```
class Id {
private static int n=0;
private int i;
public Id() {i=n; n=n+1;}
public Id(Id *x) {i=x->i; }
public print() { stampa x_i; }
public bool eq(Id *x) {return(i == x->i);}
}
```

Esercizio OOP.2

```
abstract class Term {
public abstract print();
public abstract Term sub(Sub S);
}

class Var extends Term {
private Id y;
public Var (Id x) {y=x;}
public print() {y->print();}
public Term sub(Sub S) {return(S->fetch(y));}
}

class App extends Term {
private Term s1,s2;
public App (Term t1, Term t2) {s1=t1; s2=t2;}
public print() {s1->print(); s2->print();}
public Term sub(Sub S) {return(new App(s1->sub(S),s2->sub(S)));}
}

class Sub {
private (Id*Term) list l;
public Sub() {l=nil;}
public Term fetch(Id x) {if (x,t) in l then return(t) else return(new Var(x));}
public update(Id x,Term t) {l=(x,t)::l;}
}
```

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

Esercizio FP.1

1. $\lambda x, y.yx$ ha tipo piu' generale $X \to (X \to Y) \to Y$
2. $\lambda x, y.y(yx)$ ha tipo piu' generale $X \to (X \to X) \to X$
3. $\lambda x.cons(x, x)$ non e' tipabile, perche' X e $X\ list$ non sono unificabili
4. (opzionale per lode) let $f = \lambda x.cons(x, nil)$ in $f\ f$ alla variabile f viene assegnato schema di tipo $\forall X.X \to X\ list$, mentre $f\ f$ (e l'espressione let) hanno tipo $(X \to X\ list)\ list$

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

Esercizio FP.2 Si consideri il tipo di dato albero, quindi definire la funzione $frontier$ che restituisce la lista delle foglie visitando un albero da sinistra.

```
datatype 'X tree = leaf of 'X | node of ('X tree) list;
fun frontier (leaf x) = [x]
  | frontier (node l) = concat (map frontier l);
```

dove $map : (X \to Y) \to X\ list \to Y\ list$ e $concat : (X\ list)\ list \to X\ list$ sono le seguenti funzioni

$$\begin{aligned} map\ f\ [x_1, \ldots, x_m] &= [f(x_1), \ldots, f(x_m)] \\ concat\ [l_1, \ldots, l_m] &= l_1 @ \ldots @ l_m \end{aligned}$$

Esercizio FP.3

```
fun Tmap2 f (leaf x,leaf y) = leaf(f(x,y))
  | Tmap2 f (node l1,node l2) = node (Lmap2 (Tmap f) (l1,l2))
  | Tmap2 f (_,_) = raise Match;
```