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

Si puo' usare una sintassi sia C++ che Java-like. 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. Si consiglia di usare Java, ma se si utilizza C++ trascurare i problemi legati alla mutua ricorsione tra tipi/classi e al passaggio dei parametri.

Esercizio OOP.1 (5 punti) Si definisca una classe Id per gli identificatori, con due modalita' per creare un nuovo oggetto della classe (cioe' il costruttore della classe e' overloaded). Inoltre si vuole dei metodi per stampare il nome dell' identificatore e per confrontarlo con il nome di un'altro identificatore.

```
class Id {
...
int i; nome dell'indentificatore
public:
Id() crea un Id con i diverso da quello di Id precedentemente creati
Id(Id x) crea un identificatore uguale ad x
print() {stampa x_i; } /* NON E' DA IMPLEMENTARE */
bool eq(Id x) controlla se i e' uguale alla i di x
}
```

Esercizio OOP.2 (10 punti) Si definisca una classe Sub per le sostituzioni ed una classe *astratta* $Term$ per i termini. $Term$ puo' avere varie sottoclassi, ma per semplicita' si considerano solo le sottoclassi Var per variabili e App per applicazioni.

$$\begin{aligned} x \in Id &::= \ldots \\ t \in Term &::= Var(x) \mid App(t_1, t_2) \end{aligned}$$

```
class Term {
public:
print()         stampa il termine (da implementare)
Term sub(Sub S) istanzia il termine con la sostituzione S (da implementare)
}
class Var: public Term {
Var (Id x)
print()         implementazione
Term sub(Sub S) implementazione
}
class App: public Term {
App (Term t1, Term t2)
print()         implementazione
Term sub(Sub S) implementazione
}
class Sub {
private:
(Id * Term) list S; lista associativa
public:
Sub()  sostituzione identita'
Term fetch(Id x)    restituisce il termine S(x)
update(Id x,Term t) aggiorna la sostituzione in modo che S(x)=t
}
```

Una sostituzione S la si puo' pensare come una funzione $S : Id \to Term$. Quindi la sostituzione identita' e' funzione $S(x) = Var(x)$, mentre $update(S, x, t)(y) =$ if $x = y$ then t else $S(y)$. Supponiamo di implementare una sostituzione mediante una lista associativa $(Id * Term)\ list$.

Linguaggi di Programmazione (III Anno), Febbraio 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. $\lambda x, y.yx$
2. $\lambda x, y.y(yx)$
3. $\lambda x.cons(x, x)$
4. (opzionale per lode) let $f = \lambda x.cons(x, nil)$ in ff

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 definire la funzione $frontier$ che restituisce la lista delle foglie visitando un albero da sinistra a destra.

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

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

Esercizio FP.3 (5 punti) Si consideri la seguente funzione polimorfa definita sulle liste

```
fun Lmap2 f (nil,nil) = nil
  | Lmap2 f (x::l1,y::l2) = f(x,y)::(map2 f (l1,l2))
  | Lmap2 f (_,_) = raise Match;
```

t.c. $map2\ f\ ([x_i|i \in m], [y_i|i \in m]) = [f(x_i, y_i)|i \in m]$, altrimenti (cioe' se le liste sono di lunghezza diversa) solleva l'eccezione $Match$.
Per analogia con $Lmap2$ si definisca una una funzione $Tmap2$, cioe' $Tmap2\ f\ (t_1, t_2)$ solleva l'eccezione $Match$ se t_1 e t_2 hanno *struttura* diversa, altrimenti restituisce l'albero t con la stessa struttura e t.c. $frontier(t) = Lmap2\ f\ (frontier\ t_1, frontier\ t_2)$.
Si possono considerare funzioni ausiliarie sulle liste, p.e. $Lmap2$.