Laurea Magistrale in INFORMATICA
Principi di Linguaggi di Programmazione
Paradigmi di Programmazione

prof. M. Bellia
Appello VI - 6 settembre 2011

(tempo a disposizione: 1:30' ore – Totalizzare almeno 4 sul primo – 5 sul secondo – 5 sul terzo esercizio)

Esercizio 1. (punti 10) Si consideri il seguente codice di un linguaggio a blocchi con scoping statico:

```
{int u = 2; int w[5] = {1,2,3,4,5}; int v = 0;
  void P(? int x, ?? int y){
        printf("inP:", x, y, u, v);
        y = u + x; w[v] = x+y;
        printf("outP:", x, y, u, v);}
  void Q(???int z){
        int w[4] = {u,0,v,z};
        P(w[u],u);}
  P(w[v],v); Q(w[v]);}
```

(a) (punti 6) Si assuma scoping statico:
(2) (punti 3) Si rimpiazzino gli identificatori con le coppie, (link di catena statica, posizione), ovvero Err se indefinito.
(3) (punti 3) Si dica cosa dovrebbe stampare l'esecuzione allorchè la trasmissione sia: (a) ?=name, ??=reference, ???=value; (b) ?=reference, ??=reference, ???=value
(b) (punti 4) Si assuma scoping dinamico e si dica cosa dovrebbe stampare l'esecuzione allorchè la trasmissione sia: (a) ?=name, ??=reference, ???=value; (a) ?=reference, ??=reference, ???=value

Esercizio 2. (punti 10) Si consideri la seguente definizioni Caml:

```
let rec filter = fun p l -> …
```

(a) (pt.3) La si completi in modo tale che, applicata ad un predicato p e a una lista di elementi l, calcoli la lista risultante dalla concatenazione degli elementi, nell'ordine in cui occorrono, per i quali il predicato calcola true.
(c) (pt.3) Si dia una versione tail-recursive, filterT, della funzione filter. Si mostri poi come è espressa la sua applicazione ad una lista l ed un predicato p.
(b) (pt.4) Utilizzando le sole funzioni List.fold_right, if_then_else e ::, si dia una versione iterativa, filterI, della funzione filter. Si mostri poi come è espressa la sua applicazione ad una lista l ed un predicato p.

Esercizio 3. (punti 10)
Sia Tree<…> una class Java per un tipo di dato astratto Tree che definisce alberi, finiti, con nodi interni, di arbitrario outdegree, ed etichettati con valori di tipo A. Le foglie sono etichettate con valori di un supertipo B del tipo A.
(a) (pt.4) Si dia una definizione di tale classe e la si usi per costruire il seguente albero (dove a1, a2, a3 siano valori di tipo A e b1, b2 valori di un secondo tipo B:

<a1 - <a2 -><b1 -><a3 ->>

(b) (pt.6) Si estenda Tree in una classe Tree2<…> avente anche il seguente metodo:
LinkedList<Integer> cammino(B x) – Se x etichetta un nodo dell'albero allora calcola il cammino dalla radice al primo nodo, con visita depth-first, dell'albero. Altrimenti calcola null.
Si completi l'intestazione e si mostri la definizione della nuova classe.

Esercizio 1: Soluzione
(A.2)
(Nota: entrambi gli indici partono da 0, indicando 0-reLinking e 0-offset, rispettivamente)

```
{int u = 2; int w[5] = {1,2,3,4,5}; int v = 0;
```

```
  void P(? int x, ?? int y){
        printf("inP:", (0,0), (0,1), (1,0), (1,2));
        (0,1) = (1,0) + (0,0); (1,1)[(1,2)] = (0,0)+(0,1);
        printf("outP:", (0,0), (0,1), (1,0), (1,2));}
  void Q(???int z){
        int w[4] = {(1,0), 0, (1,2), (0,0)};
        (1,3)((0,1)[(1,0)],(1,0));}
  (0,3)((0,1)[(0,2)],(0,2)); (0,4)((0,1)[(0,2)]);}
```

(a.3)
(Nota: indici array partono da 0)

a: inP: 1 0 2 0
outP: 7 3 2 3
inP: 3 2 2 3
--- errore per array outOfbound ---

b: inP: 1 0 2 0
outP: 1 3 2 3
inP: 3 2 2 3
outP: 3 5 5 3

(B)
Siano nel caso (a) che nel caso (b) la stampa coincide (Non lo stato della memoria), per cui non la ripetiamo.

Esercizio 2: Soluzione
(a)

```
let rec filter = fun p l -> match l with
        |[] -> []
        |x::lR -> if (p x) then x::(filter p lR) else (filter p lR);;
filter p l;;
```

(b)

```
let rec filterT = fun p l r -> match l with
        |[] -> r
        |x::lR -> filterT p lR (if (p x) then r@[x] else r);;
filterT p l [];;
```

(c)

```
let filterI = fun p l -> List.fold_right (fun x -> fun u -> if (p x) then (x::u) else u) l [];;
filterI p l;;
```

Esercizio 3: Soluzione (progettato per valori non modificabili. Il caso modificabile inutilmente più complicato)
(A)

```
class Tree <B, A extends B> {
                private B nodo;
                private LinkedList<Tree<B,A>> figli;
        private boolean interno;
        public Tree(B x){
                nodo = x;
                interno = false;}
        public Tree(A x, LinkedList<Tree<B,A>> s){
                /* s.size() deve essere definito e diverso da 0 */
                if (s==null) {nodo=x; interno=false; return;}
                nodo = x;
                figli = s;
                interno = true;}
        public B etichetta(){return nodo;};
        public LinkedList <Tree<B,A>> figli(){return figli;};
}
/*costruzione dell'albero*/
class B{};
class A extends B{};
...
public static void main (String [] args){
        ...
        B b1 = new B();
        A a1 = new A();
        A a2 = new A();
        A a3 = new A();
        LinkedList<Tree<B,A>>temp = new LinkedList<Tree<B,A>>();
        temp.add(new Tree<B,A>(a2));
        temp.add(new Tree<B,A>(b1));
        temp.add(new Tree<B,A>(a3));
```

```
		Tree<B,A> res = new Tree<B,A>(a1,temp);
(B)
class Tree2<B, A extends B> extends Tree<B,A>{
		public Tree2(B x){
			super(x);}
		public Tree2(A x, LinkedList<Tree<B,A>> s){
			super(x,s);}
		public LinkedList<Integer> cammino(B x){
			LinkedList<Integer> res  = new LinkedList<Integer>();
			if (x.equals(etichetta())){res.add(0); return res;}
			LinkedList<Tree<B,A>> f = figli();
			if (f == null) return null;
			for (int i=0; i<f.size();i++){
				res = ((Tree2<B,A>)f.get(i)).cammino(x);
				if (!(res == null) && (res.getLast() == 0)) {res.addFirst(i+1); return res;}}
			return null;}
}
```