Appello II - 6 febbraio 2012:
Paradigmi: SOLUZIONE

Esercizio1
(a) È corretto. Stampa: "valore di z = 3".
(b)
```
{int z = 3;
 while ([0,0] > 3){
        printf("valore di z = %2d\n", [1,0]);
        int [0,0] = [0,0] -1;
        printf("valore di z = %2d\n", [0,0]);
        }
 printf("valore di z = %2d\n", [0,0]);}
```

Esercizio2
(a)
```
module type GRAPH =
 sig type ('a,'b) graph
  val mkG: 'a -> 'b -> ('a,'b) graph
  val addN: 'a node -> ('a,'b) graph -> ('a,'b) graph
  val addE: 'a node -> 'a node -> ('a,'b) graph -> 'b -> ('a,'b) graph
  val nodes: ('a, 'b) graph -> ('a node) list
  val outN: 'a node -> ('a,'b) graph -> ('a node) list
end;;
```
dove assumiamo queste definizioni esterne per nodo e arco:
```
type 'a node = N of 'a * int
type ('a,'b) edge = E of 'a node * 'a node * 'b
```
(b)
```
module GRAPH1 =
 (struct
  type ('a,'b) outer = C of 'a node * (('a,'b) edge) list
  type ('a,'b) graph = G of 'a * 'b * (('a,'b) outer)list
  let mkG na eb = G(na,eb,[])
  let addN na g = let ol = getE(g) in
    if (isNin na ol) then raise (Failure "addN")
        else G(getLN(g),getLE(g),C(na,[])::ol)
  let addE ns nt g lb = ...
  let isNin na g = ...
  let isEin ns nt lb g = ...
  let nodes (G(na,nb,ol)) = List.map outerN ol
  let outN (ns,g) = let ol = (getE g) in
    if not (isNin ns ol) then raise(Failure "outE")
            else List.fold_right(@)(List.map (fun x -> if eqN((outerN x),ns)
                    then (outerL x) else []) ol)[]
 (*metodi ausiliari:inizio*)
  let outerN (C(n,_)) = n
  let outerL (C(_,l)) = l
  let getLN (G(labN,labE,edges)) = labN
  let getLE (G(labN,labE,edges)) = labE
  let getE (G(labN,labE,edges)) = edges
  let eqN(N(u,n1),N(v,n2)) = (u==v)&&(n1==n2)
end:GRAPH);;
```

Esercizio3
(a)
```
public class GRAPH<A,B> {
    ...
    GRAPH(){...}
    public GRAPH<A,B> mkG(A x,B y){...}
    public void addN(Node<A> na)throws FailureException{...}
    public void addE(Node<A> ns, Node<A> nt, B lb) throws FailureException{...}
    public boolean isNin(Node<A> x){...}
```

```
    public boolean isEin(Edge<A,B> e){...}
    public LinkedList<Node<A>> nodes() {...}
    public LinkedList<Node<A>> outN(Node<A> ns) throws FailureException{...}
}
(b)
class AGRAPH <A,B> extends GRAPH <A,B>{
    private Vector<Node<A>> indegree;
    AGRAPH(){indegree = new Vector<Node<A>>();}
    public void addE(Node<A> ns, Node<A> nt, B lb) throws FailureException{
        Edge<A,B> e = new Edge<A,B>(ns,nt,lb);
        if (isEin(e)) return;
        if (indegree.contains(nt)) return;
        indegree.add(nt);
        super.addE(ns,nt,lb);}
}
(c)
class BGRAPH <A,B> extends AGRAPH <A,B>{
    public LinkedList<Node<A>> reach(Node<A> ns) throws FailureException{
        LinkedList<Node<A>> r = outN(ns);
        LinkedList<Node<A>> newfound = outN(ns);
        while(newfound.size()!=0){
            LinkedList<Node<A>> temp = new LinkedList<Node<A>>();
            for(Node<A> u : newfound)temp.addAll(diff(outN(u),r));
            r.addAll(temp); newfound=temp;}
        return r;}
    private static<A> LinkedList<Node<A>> diff(LinkedList<Node<A>>t, LinkedList<Node<A>>s){
        LinkedList<Node<A>> r = new LinkedList<Node<A>>();
        for(Node<A> u : t){if(!s.contains(u))r.add(u)}
        return r;}
}
```