Corso di Paradigmi di Programmazione
Prova scritta del 23 Settembre 2003.

Tempo a disposizione: ore 2.

1. Si illustri brevemente la gestione dinamica della memoria mediante heap, specificando in quali casi tale tipo di gestione e' necessario.

2. Si fornisca un esempio di un insieme di dichiarazioni di procedura e di una sequenza di chiamate (in un qualsiasi pseudo-linguaggio) tali che la catena statica e quella dinamica risultanti siano diverse.

3. Si fornisca una grammatica che genera il linguaggio $\{a^n b^m | n, m \geq 0\}$

4. Si dica cosa viene stampato dal seguente frammento di codice scritto in uno pseudo-linguaggio che usa scoping dinamico e shallow binding:

```
x: integer;

procedure ass_x(n:integer)
    x = n;

procedure stampa_x
    write_integer(x);

procedure pippo(S, P: function; n: integer)
    x: integer;
    if n=1  then ass_x(n)
            else S(n);
    P;

ass_x(0);
pippo(ass_x, stampa_x, 1);
stampa_x; ass_x(0);
pippo(ass_x, stampa_x, 2);
stampa_x;
```

5. Si dica cosa viene stampato dal seguente frammento di codice scritto in uno pseudo-linguaggio che ammette parametri per *riferimento*.

```
int X[10];
int i = 1;
X[0] = 0;
X[1] = 0;
X[2] = 0;
void foo (reference int Y,J){
     X[J] = J+1;
     write(Y);
     J++;
     X[J]=J;
     write(Y);
     }
foo(X[i],i);
write(X[i]);
```

6. Si descriva la tecnica delle *tombstones*, discutendo sinteticamente il problema che tale tecnica affronta e i principali svantaggi.

7. Si supponga di avere le seguenti classi Java:

```
class A { int num;}

class C {void foo(A a){ a.num++;}
                void fie (int n) { n++;}
        }
```

e di eseguire ora il codice che segue:

```
int n=0;
A a = new A();
A a2 = a;
a.num=0;
System.out.print(n);
System.out.print(a.num);
System.out.print(a2.num);
C c = new C();
c.foo(a);
c.fie(n);
System.out.print(n);
System.out.print(a.num);
System.out.print(a2.num);
```

Si dica cosa viene stampato. È noto che Java permette il solo passaggio per valore. Come dunque devono essere spiegati i risultati della stampa?

8. Sono date le seguenti definizioni in Java:

```
interface A {
   void foo();
}

interface B extends A {
    void fieB();
}

interface C extends A {
    void fieC();
}

class D {
    void pippo(){System.out.print("pippo");}
}
```

Viene adesso introdotta la seguente classe:

```
class E extends D implements B,C {
    public void fieB(){}
    public void fieC(){}
    ...
}
```

Si dia del codice (il più semplice possibile) da inserire al posto dei puntini (...) affinché la classe `E` costituisca una definizione legale.

Quali *tipi* introducono le definizioni date (da `A` a `E`)?

Si descriva graficamente la gerarchia di sottotipo esistente tra questi tipi.