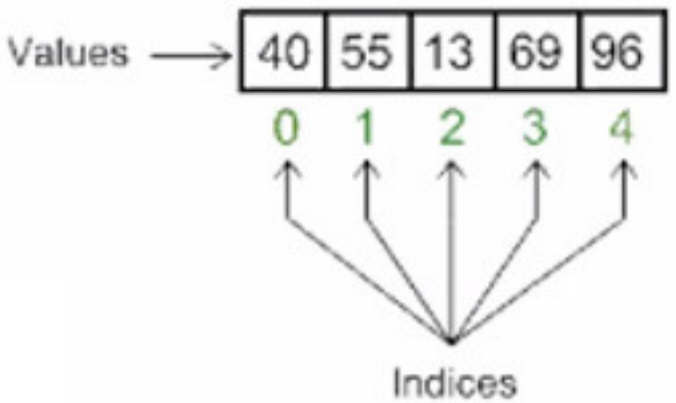
```
class Nodo{
private:
  Nodo(char c='*', Nodo* s=0, Nodo* d=0): info(c), sx(s), dx(d) {}
  char info;
  Nodo* sx;
  Nodo* dx;
};
class Tree {
public:
  Tree(): root(0) {}
  Tree(const Tree&); // dichiarazione costruttore di copia
private:
  Nodo* root;
};
```

Gli oggetti della classe Tree rappresentano alberi binari ricorsivamente definiti di char. Si ridefiniscano assegnazione, costruttore di copia e distruttore di Tree come **assegnazione, copia e distruzione profonda**. Scrivere esplicitamente eventuali dichiarazioni friend che dovessero essere richieste da tale definizione.

```
/* ESERCIZIO:
Definire una classe vettore i cui oggetti rappresentano array di interi.
vettore deve includere un costruttore di default, una operazione di
concatenazione che restituisce un nuovo vettore v1+v2, una operazione di
append v1.append(v2), l'overloading dell'uguaglianza, dell'operatore di
output e dell'operatore di indicizzazione.  Deve inoltre includere il
costruttore di copia profonda, l'assegnazione profonda e la distruzione profonda.
*/

#include<iostream>

class Vettore {
private:
  int* a;
  unsigned int size; // size ≧ 0  (garantito da unsigned int)
  // vettore vuoto IFF a==nullptr && size == 0
  // vettore non vuoto IFF a!=nullptr && size>0
public:
  // unsigned int => Vettore
  Vettore(unsigned int s =0, int init=0): a(s==0 ?  nullptr : new int[s]), size(s) {
    for(int j=0; j<size; ++j) a[j]=init;
  }

  Vettore(const Vettore& v): a(v.size == 0 ? nullptr : new int[v.size]), size(v.size) {
    for(unsigned int j=0; j<size; ++j) a[j]=v.a[j];
  }

  Vettore& operator=(const Vettore& v) {
    if (this != &v) {
      delete[] a; // attenzione: delete[] e NON delete
      size = v.size;
      a = size == 0 ? nullptr : new int[size];
      for (int i = 0; i < size; i++)
      a[i] = v.a[i];
    }
    return *this;
  }

  ~Vettore() {if(a) delete[] a;}
};

int main() {
  Vettore x;
  Vettore v1, v2(3), v3(5);
}
```

Definire una classe **Vettore** i cui oggetti rappresentano array di interi. Vettore deve includere un costruttore di default, una operazione di concatenazione che restituisce un nuovo vettore v1+v2, una operazione di append v1.append(v2), l'overloading dell'uguaglianza, dell'operatore di output e dell'operatore di indicizzazione. Deve inoltre includere il costruttore di copia profonda, l'assegnazione profonda e la distruzione profonda.



```
/*
Gli oggetti della classe Tree rappresentano alberi binari ricorsivamente definiti di char. Si
ridefiniscano assegnazione, costruttore di copia e distruttore di Tree come assegnazione, copia e
distruzione profonda. Scrivere esplicitamente eventuali dichiarazioni friend che dovessero essere
richieste da tale definizione.
*/

class Nodo {
  friend class Tree;
private:
  Nodo(char c='*', Nodo* s=0, Nodo* d=0): info(c), sx(s), dx(d) {}
  char info;
  Nodo* sx;
  Nodo* dx;
};

class Tree {
private:
  Nodo* root;
  static Nodo* copia(Nodo* r) {
    // caso base: albero vuoto
    if(r==nullptr) return nullptr;
    // passo induttivo: dalle copie profonde ritornate induttivamente da
    // copia(r->sx) e copia(r->dx), costruisco la copia dell'albero radicato in *r
    // nel seguente modo:
    return new Nodo(r->info,copia(r->sx),copia(r->dx));
  }
  static void distruggi(Nodo* r) {
    // caso base: albero vuoto
    if(r!=nullptr) {
      // passo induttivo: albero non vuoto
      distruggi(r->dx);
      distruggi(r->sx);
      delete r;
    }
  }

public:
  Tree(): root(nullptr) {}
  Tree(const Tree& t): root(copia(t.root)) {} // copia profonda
  Tree& operator=(const Tree& t) {
    if(this != &t) {
      distruggi(root);
      root = copia(t.root);
    }
    return *this;
  }

  ~Tree() {if(root) distruggi(root);}
};

int main() {
  Tree t1,t2;
  t1=t2;
  Tree t3=t2;
}
```