Il seguente programma compila correttamente. L'esecuzione quali stampe provoca in output su `cout`? Scrivere "VALORE CASUALE" quando si prevede che una certa istruzione provochi una stampa di un valore casuale, e "RUN-TIME ERROR" quando si prevede che una certa istruzione provochi un errore run-time.

```
class C {
public:
  int number;
  C(int n=1) : number(n) { cout << "C(" << number << ") "; }
  ~C() { cout << "~C(" << number << ") "; }
  C& operator=(const C& c) { number=c.number; cout << "operator=(" << number << ") "; }
};

int F(C c) {return c.number;}

int main() {
  C *c=new C; cout << "UNO" << endl;
  C d; d=*c; cout << "DUE" << endl;
  int x=F(d); cout << "TRE" << endl;
  int y=F(F(d)); cout << "QUATTRO" << endl;
}
```

Il seguente programma compila correttamente. L'esecuzione quali stampe provoca in output su `cout`? Scrivere "VALORE CASUALE" quando si prevede che una certa istruzione provochi una stampa di un valore casuale, e "RUN-TIME ERROR" quando si prevede che una certa istruzione provochi un errore run-time.

```
class C {
public:
  int a[2];
  C(int x=0,int y=1) {a[0]=x; a[1]=y; cout << "C(" << a[0] << "," << a[1] << ") ";}
  C(const C&) {cout << "Cc ";}
};

class D {
private:
  C c1;
  C *c2;
  C& cr;
public:
  D() : c2(&c1), cr(c1) { cout << "D() ";}
  D(const D& d) : cr(c1) { cout << "Dc ";}
  ~D() { cout << "~D ";}
};

class E {
public:
  static C cs;
};
C E::cs=1;

int main() {
  C c; cout << "UNO" << endl;
  C x(c); cout << x.a[0] << " " << x.a[1] << " DUE" << endl;
  D d=D(); cout << "TRE" << endl;
  E e;cout << "QUATTRO" << endl;
}
```

Esercizio 2.10.4. Si osservino e confrontino le stampe provocate dall'esecuzione dei seguenti due programmi.

```
class C {
public:
  C() {cout << "C0 ";}
  C(const C&) {cout << "Cc ";}
};

class D {
public:
  C c;
  D() {cout << "D0 ";}
};

int main() {
  D x; cout << endl;
  // stampa: C0 D0
  D y(x); cout << endl;
  // stampa: Cc
}
```

```
class C {
public:
  C() {cout << "C0 ";}
  C(const C&) {cout << "Cc ";}
};

class D {
public:
  C c;
  D() {cout << "D0 ";}
  D(const D&) {cout <<"Dc ";}
};

int main() {
  D x; cout << endl;
  // stampa: C0 D0
  D y(x); cout << endl;
  // stampa: C0 Dc
}
```

Si noti attentamente che la differenza è spiegata dal fatto che il costruttore di copia standard di una classe `D` invoca ordinatamente per ogni campo dati `x` di `D` il corrispondente costruttore di copia del tipo di `x`.

Il seguente programma compila ed esegue correttamente. Quali stampe produce in output la sua esecuzione?

```
class A {
  friend class C;
private:
  int k;
public:
  A(int x=2): k(x) {}
  void m(int x=3) {k=x;}
};

class C {
private:
  A* p;
  int n;
public:
  C(int k=3) {if (k>0) {p = new A[k]; n=k;}}
  A* operator->() const {return p;}
  A& operator*() const {return *p;}
  A* operator+(int i) const {return p+i;}
  void F(int k, int x) {if (k<n) p[k].m(x);}
  void stampa() const {
    for(int i=0; i<n; i++) cout << p[i].k << ' ';
  }
};

int main() {
  C c1;  c1.F(2,9);
  C c2(4); c2.F(0,8);
  *c1=*c2;
  (c2+3)->m(7);
  c1.stampa(); cout << "UNO\n";
  c2.stampa(); cout << "DUE\n";
  c1=c2;
  *(c2+1)=A(3);
  c1->m(1);
  *(c2+2)=*c1;
  c1.stampa(); cout << "TRE\n";
  c2.stampa(); cout << "QUATTRO";
}
```

Si considerino le seguenti dichiarazioni e definizioni:

```
class Nodo {
private:
  Nodo(string st="***", Nodo* s=0, Nodo* d=0): info(st), sx(s), dx(d) {}
  string info;
  Nodo* sx;
  Nodo* dx;
};
class Tree {
public:
  Tree(): radice(0) {}
  Tree(const Tree&); // dichiarazione costruttore di copia
private:
  Nodo* radice;
};
```

Quindi, gli oggetti della classe `Tree` rappresentano *alberi binari ricorsivamente definiti di stringhe*. Si ridefinisca il costruttore di copia di `Tree` in modo che esegua copie profonde. Scrivere esplicitamente eventuali dichiarazioni `friend` che dovessero essere richieste da tale definizione.