Esercizi di Programmazione ad Oggetti

Lista n. 2

Esercizio 1

Il seguente programma compila.

```
class S {
public:
  string s;
  S(string t): s(t) {}
};
class N {
private:
  S x;
public:
  N* next;
  N(S t, N* p): x(t), next(p) {cout << "N2 ";}
  ~N() {if (next) delete next; cout << x.s + "~N ";}
};
class C {
  N* pointer;
public:
  C(): pointer(0) {}
  ~C() {delete pointer; cout << "~C ";}
  void F(string t1, string t2 = "pippo") {
    pointer = new N(S(t1),pointer); pointer = new N(t2,pointer);
  }
};
main(){
  C* p = new C; cout << "UNO\n";
  p->F("pluto","paperino"); p->F("topolino"); cout <<"DUE\n";
  delete p; cout <<"TRE\n";
}
```

Soluzione.

```
NESSUNA STAMPA UNO
N2 N2 N2 N2 DUE
pluto~N paperino~N topolino~N pippo~N ~C TRE
NESSUNA STAMPA
```

Esercizio 2

Si consideri il seguente programma.

```
#include<iostream>
using std::cout;

class It {
  friend class C;
public:
  bool operator<(It i) {return index < i.index;}
  It operator++(int) { It t = *this; index++; return t; }
  It operator+(int k) {index = index + k; return *this; }
private:
  int index;
};
class C {
public:
  C(int k) {
    if (k>0) {dim=k; p = new int[k];}
    for(int i=0; i<k; i++) *(p+i)=i;
  }
  It begin() { It t; t.index = 0; return t; }
  It end() { It t; t.index = dim; return t; }
  int& operator[](It i) {return *(p + i.index);}
private:
  int* p;
  int dim;
};

main() {
  C c1(4), c2(8);
  for(It i = c1.begin(); i < c1.end(); i++) cout << c1[i] << ' ';
  cout << "UNO\n";
  It i = c2.begin();
  for(int n=0; i < c2.end(); ++n, i = i+n) cout << c2[i] << ' ';
  cout << "DUE\n";
}
```

Il programma compila correttamente? Se sì, quali stampe provoca in output?
Soluzione.

```
0 1 2 3 UNO
0 1 3 6 DUE
```

Esercizio 3

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.
Soluzione.

```
class Tree {
public:
  Tree(): radice(0) {}
  Tree(const Tree& t) {
    radice = copia(t.radice);
  }
private:
  Nodo* radice;
  static Nodo* copia(Nodo* p) {
    if (!p) return 0;
    else return new Nodo(p->info,copia(p->sx),copia(p->dx));
  }
};
```

Esercizio 4

Definire una classe `Vettore` i cui oggetti rappresentano array di interi. `Vettore` deve includere un costruttore di default, l'overloading dell'uguaglianza, dell'operatore di output e dell'operatore di indicizzazione. Inoltre deve includere un costruttore di copia "profonda" e l'overloading dell'assegnazione come assegnazione "profonda".
Soluzione.

```
// vettore.h
#ifndef VETTORE_H
#define VETTORE_H
#include<iostream>
using std::ostream;

class Vettore {
public:
  Vettore(int =1, int =0);
  Vettore(const Vettore&);
  Vettore& operator=(const Vettore&);
  int& operator[](int) const; //operatore di indicizzazione
  bool operator==(const Vettore&) const;
  int dim() const { return size; } // inline
private:
  int size; // dimensione
  int* v; // array dinamico
};

ostream& operator<<(ostream&, const Vettore&);
#endif
```

```
// vettore.cpp
#include "Vettore.h"
using std::cout; using std::endl;

Vettore::Vettore(int sz, int x): size(sz), v(new int[sz]) {
  for(int i=0; i<size; i++) v[i] = x;
}

Vettore::Vettore(const Vettore& x): size(x.size), v(new int[x.size]) {
  for(int i=0; i<size; i++) v[i]=x.v[i];
}

Vettore& Vettore::operator=(const Vettore& x) {
  if(this != &x) { // se siamo nel caso x=x non faccio nulla
    delete[] v; // dealloco
    size = x.size;
    v = new int[x.size];
    for(int i=0; i<size; i++) v[i]=x.v[i];
  }
  return *this;
}

int& Vettore::operator[](int i) const { return v[i]; }

bool Vettore::operator==(const Vettore& x) const {
  if(size != x.size) return false;
  for(int i=0; i<size; i++)
    if(v[i] != x.v[i]) return false;
```

```
  return true;
}

ostream& operator<<(ostream& ostr, const Vettore& x) {
  ostr << '(';
  for(int i=0; i<x.dim()-1; i++){
    ostr << x[i] <<',';
    if((i+1)%10 == 0) cout << endl;
  }
  return ostr << x[x.dim()-1] << ")\n";
}
```

Esercizio 5

```
class C {
public:
    C(): size(1), a(new int[1]) {a[0]=0;}
    C& operator=(const C& x) {
          if(this!=&x){
            size=x.size;
            a=new int[size];
            for(int i=0;i<size;i++) a[i]=x.a[i];
          }
          return *this;
    }
    void add(int k) {
      int *b=a;
          a=new int[size+1];
          ++size;
          a[0]=k;
          for(int i=1;i<size;i++) a[i]=b[i-1];
          delete[] b;
    }
    int& operator[](int i) const {return a[i];}
    void stampa() const {
          for(int i=0;i<size;i++) cout<<a[i]<< ' ';
    }
    ~C() {stampa(); cout<<"~C "; delete[] a;}
private:
    int size;
    int* a;
};

main(){
    C v; v.add(1);
    C w=v; w[1]=2;
    v.stampa(); cout<<"UNO\n";
    w.stampa(); cout<<"DUE\n";
    C* p=new C; p->add(3);
    *p=v;
    (*p)[0]=4; v[1]=5;
    v.stampa(); cout<<"TRE\n";
    w.stampa(); cout<<"QUATTRO\n";
    p->stampa(); cout<<"CINQUE\n";
    w=*p;
    w[1]=6; v[0]=7;
    v.stampa(); cout<<"SEI\n";
    w.stampa(); cout<<"SETTE\n";
    p->stampa(); cout<<"OTTO\n";
    delete p; cout<<"NOVE\n";
}
```

Il precedente programma compila correttamente. Quali stampe provoca la sua esecuzione?
Soluzione.

```
1 2 UNO
1 2 DUE
1 5 TRE
1 5 QUATTRO
4 2 CINQUE
7 5 SEI
```

```
4 6 SETTE
4 2 OTTO
4 2 ~C NOVE
4 6 ~C 7 5 ~C
```