Programmazione 2
Appello d'esame – 14/7/2003

Nome..... Cognome.....

Matricola..... Laurea in.....

Tempo a disposizione: 2H30M**. Non si possono consultare appunti e libri. Dove previsto scrivere CHIARAMENTE la risposta nell'apposito spazio.**

1. Il seguente programma compila.

```
#include<iostream>
#include<vector>

class C {
public:
  C(int y = 5): v(vector<int>()) {v.push_back(y); cout << "C(int) ";};
  C(const C& x) {v=x.v; cout << "Cc ";};
  ~C() {cout << "~C() ";};
  int lung() {return v.size();}
protected:
  vector<int> v;
};

class D: public C {
public:
  D(int j = 2): z(j+v.size()) {v.push_back(z); cout << "D(int) ";};
  D(const D& x): C(x) {z = x.z; cout << z << " Dc ";};
  ~D() {cout << z << " ~D() ";};
private:
  int z;
};

D Fun(C x, D& y) {
  cout << "Pippo ";
  int k = 2*x.lung() + y.lung();
  D d(k);
  cout << "Pluto ";
  return d;
};

main() {
  C c(3); cout << "UNO\n";
  D d; cout << "DUE\n";
  D d1 = Fun(c,d); cout << "TRE\n";
  d1 = Fun(d,d1); cout << "QUATTRO\n";
}
```

Quali stampe produce la sua esecuzione? Se una istruzione non produce alcuna stampa si scriva **NESSUNA STAMPA**. Si noti che ci possono essere delle stampe anche dopo "QUATTRO".

..... UNO

..... DUE

..... TRE

..... QUATTRO

.....

2. Il seguente programma compila.

```
#include<iostream>
#include<string>
#include<typeinfo>

class Z {
public:
  Z() {cout << "Z() ";};
  Z(int x) {cout << "Z(int) ";};
};

class A {
protected:
  int z;
public:
  A() {z=0; cout << "A() ";};
  A(int x, int y) {z=x+y; cout << "A(int,int) ";};
  virtual ~A() {};
};

class B: virtual public A {
protected:
  static int s;
public:
  B() {cout << s << " B() ";};
  B(int k): A(k,k) {z+=k; cout << z << " B(int) ";};
};
int B::s = 1;

class C: virtual public A {
protected:
  Z* a;
public:
  C(int n = 1) {a = new Z[n]; cout << "C0-1 ";}
};

template<class T>
class D: virtual public C {
protected:
  T a;
public:
  D() {cout << "D() ";};
  D(int x, T y): a(x), C(x) {cout << x << " D(int," << typeid(T).name() << ") ";};
};

class E: virtual public C {
public:
  E(): C(1) {z++; cout << "E() ";};
};

template<class T>
class F: public B, public D<T>, public E {
public:
  F(): A(0,1), B(2) {s++; cout << s << " F() ";};
  F(T x): D<T>(2,x) {cout << z << " F(" << typeid(x).name() << ") ";};
};
```

```
main() {
  Z z; cout << "UNO\n";
  D<B> d; cout << "DUE\n";
  C* pc = &d; cout << "TRE\n";
  F<Z> f1(z); cout << "QUATTRO\n";
  F<string> f2; cout << "CINQUE\n";
  D<Z> d1(2,z); cout << "SEI\n";
  E e(f2); cout << "SETTE\n";
  C a[3] = {C(2), E(), C() }; cout << "OTTO\n";
}
```

Quali stampe produce la sua esecuzione? Se una istruzione non produce alcuna stampa si scriva **NESSUNA STAMPA**.

..... `UNO`

..... `DUE`

..... `TRE`

..... `QUATTRO`

..... `CINQUE`

..... `SEI`

..... `SETTE`

..... `OTTO`

3. Sia `B` una classe con distruttore pubblico e virtuale e sia `C` una sottoclasse di `B`. Definire una funzione `int Fun(vector<B*>& v)` con il seguente comportamento: sia `v` non vuoto e sia `T*` il tipo dinamico di `v[0]`; allora `Fun(v)` ritorna il numero di elementi di `v` che hanno un tipo dinamico `T1*` tale che `T1` è un sottotipo di `C` diverso da `T`; se `v` è vuoto deve quindi ritornare `0`.
   Ad esempio, il seguente programma deve compilare e provocare le stampe indicate.

```
#include<iostream>
#include<typeinfo>
#include<vector>

class B {public: virtual ~B() {} };
class C: public B {};
class D: public B {};
class E: public C {};

int Fun(vector<B*> &v){...}

main() {
  vector<B*> u, v, w;
  cout << Fun(u); // stampa 0
  B b; C c; D d; E e; B *p = &e, *q = &c;
  v.push_back(&c); v.push_back(&b); v.push_back(&d); v.push_back(&c);
  v.push_back(&e); v.push_back(p);
  cout << Fun(v); // stampa 2
  w.push_back(p); w.push_back(&d); w.push_back(q); w.push_back(&e);
  cout << Fun(w); // stampa 1
}
```

4. Il seguente programma compila.

```
#include<iostream>

class A {
public:
  int z;
  A(int x=1): z(x) {cout << "A(" << x <<") ";};
};

template<class T>
class C {
public:
  static A s;
};
template<class T>
A C<T>::s=A();

main() {
  C<double> c;
  C<bool>::s.z = 4;
  C<int> d;
  C<int>::s = A(2);
  C<double>::s = C<int>::s;
}
```

   Quale stampa produce la sua esecuzione? Se non produce alcuna stampa si scriva **NESSUNA STAMPA**.

   .....

5. Si considerino le seguenti definizioni:

```
class nodo {
public:
  nodo();
  nodo(int x, nodo* n, nodo* p);
  int info;
  nodo* next;
  nodo* prec;
};

class lista {
public:
  lista();
  lista(const lista&);
  bool Vuota();
  void Aggiungi_In_Testa(int);
  void Aggiungi_In_Coda(int);
  lista& operator=(const lista&);
private:
  nodo* first, last;
};
```

   Gli oggetti della classe `lista` rappresentano liste di interi. Un oggetto della classe `nodo` rappresenta un nodo di tali liste, ed ogni nodo contiene sia un puntatore `next` al nodo successivo che un puntatore `succ` al nodo precedente. La rappresentazione delle liste prevede un puntatore `first` al primo nodo ed un puntatore `last` all'ultimo nodo. Si richiede l'implementazione dei costruttori e metodi delle classi `nodo` e `lista` in modo che **non vi sia condivisione di memoria** tra gli oggetti

della classe `lista`.
Ad esempio, il seguente `main()` (si suppone che sia stato opportunamente ridefinito `operator<<`) deve compilare e provocare le stampe indicate.

```
main() {
  lista l1;
  l1.Aggiungi_In_Testa(3); l1.Aggiungi_In_Coda(4);
  cout << l1; // stampa: 3 4
  lista l2(l1);
  l2.Aggiungi_In_Testa(5);
  cout << l1; // stampa: 3 4
  cout << l2; // stampa: 5 3 4
  lista l3; l3.Aggiungi_In_Coda(6);
  cout << l3; // stampa: 6
  l3 = l2;
  l3.Aggiungi_In_Coda(7);
  cout << l2; // stampa: 5 3 4
  cout << l3; // stampa: 5 3 4 7
}
```