Programmazione 2
Seconda Prova Parziale – 1/12/2003

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

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

1. Definire una superclasse `Auto` i cui oggetti rappresentano generiche automobili e due sue sottoclassi `Benzina` e `Diesel`, i cui oggetti rappresentano automobili alimentate, rispettivamente, a benzina e a diesel (ovviamente non esistono automobili non alimentate e si assume che ogni auto è o benzina o diesel). Ci interesserà l'aspetto fiscale delle automobili, cioè il calcolo del bollo auto. Queste classi devono soddisfare le seguenti specifiche:

   - Ogni automobile è caratterizzata da un numero di cavalli fiscali. La tassa per cavallo fiscale è unica per tutte le automobili (sia benzina che diesel) ed è fissata in 5 euro. La classe `Auto` fornisce un metodo `tassa()` che ritorna la tassa di bollo fiscale per l'automobile di invocazione.
   - La classe `Diesel` è dotata (almeno) di un costruttore ad un parametro intero `x` che permette di creare un'auto diesel di `x` cavalli fiscali. Il calcolo del bollo fiscale per un'auto diesel viene fatto nel seguente modo: si moltiplica il numero di cavalli fiscali per la tassa per cavallo fiscale e si somma una addizionale fiscale unica per ogni automobile diesel fissata in 100 euro.
   - Un'auto benzina può soddisfare o meno la normativa europea Euro4. La classe `Benzina` è dotata di (almeno) un costruttore ad un parametro intero `x` e ad un parametro booleano `b` che permette di creare un'auto benzina di `x` cavalli fiscali che soddisfa Euro4 se `b` vale `true` altrimenti che non soddisfa Euro4. Il calcolo del bollo fiscale per un'auto benzina viene fatto nel seguente modo: si moltiplica il numero di cavalli fiscali per la tassa per cavallo fiscale; se l'auto soddisfa Euro4 allora si detrae un bonus fiscale unico per ogni automobile benzina fissato in 50 euro, altrimenti non vi è alcuna detrazione.

   Si definisca inoltre una classe `ACI` i cui oggetti rappresentano delle filiali ACI addette all'incasso dei bolli auto. Ogni oggetto ACI è caratterizzato da un vector di puntatori ad auto, cioè un oggetto `vector<Auto*>`, che rappresenta la lista delle automobili gestite da una filiale ACI. La classe `ACI` fornisce un metodo `aggiungiAuto(Auto& a)` che aggiunge l'auto `a` alla lista gestita dalla filiale di invocazione. Inoltre, la classe `ACI` fornisce un metodo `incassaBolli()` che ritorna la somma totale dei bolli che devono pagare tutte le auto gestite dalla filiale di invocazione.

   Definire infine un esempio di `main()` in cui viene costruita una filiale ACI a cui vengono aggiunte quattro automobili in modo tale che l'incasso dei bolli ammonti a 1600 euro.

2. Si considerino le seguenti definizioni di template di classe.

```
#include<iostream>
using namespace std;

template<class T> class D; // dichiarazione incompleta

template<class T1, class T2>
class C {
  friend class D<T1>;
private:
  T1 t1;
  T2 t2;
};

template<class T>
class D {
public:
  void m() {C<T,T> c; cout << c.t1 << c.t2;}
  void n() {C<int,T> c; cout << c.t1 << c.t2;}
  void o() {C<T,int> c; cout << c.t1 << c.t2;}
  void p() {C<int,int> c; cout << c.t1 << c.t2;}
  void q() {C<int,double> c; cout << c.t1 << c.t2;}
  void r() {C<char,double> c; cout << c.t1 << c.t2;}
};
```

Per ciascuno dei seguenti `main()` segnare con una croce (COMPILA) oppure (NON COMPILA) per indicare la corretta compilazione o meno.

`main() { D<char> d; d.m(); }`	(COMPILA)	(NON COMPILA)
`main() { D<char> d; d.n(); }`	(COMPILA)	(NON COMPILA)
`main() { D<char> d; d.o(); }`	(COMPILA)	(NON COMPILA)
`main() { D<char> d; d.p(); }`	(COMPILA)	(NON COMPILA)
`main() { D<char> d; d.q(); }`	(COMPILA)	(NON COMPILA)
`main() { D<char> d; d.r(); }`	(COMPILA)	(NON COMPILA)

3. Definire un template di funzione `Fun(T1*, T2&)` che ritorna un booleano con il seguente comportamento. Consideriamo una istanziazione implicita `Fun(p,r)` dove supponiamo che i parametri di tipo `T1` e `T2` siano istanziati a tipi polimorfi (cioè che contengono almeno un metodo virtuale). Allora `Fun(p,r)` ritorna `true` se e soltanto se valgono le seguenti condizioni:

   (a) i parametri di tipo `T1` e `T2` sono istanziati allo stesso tipo;

   (b) siano `D1*` il tipo dinamico di `p` e `D2&` il tipo dinamico di `r`. Allora (i) `D1` e `D2` sono lo stesso tipo e (ii) questo tipo è un sottotipo proprio della classe `ios` della gerarchia di classi di I/O (si ricordi che `ios` è la classe base astratta della gerarchia).

   Ad esempio, il seguente `main()` deve compilare e provocare le stampe indicate:

```
#include<iostream>
#include<fstream>
#include<typeinfo>
using namespace std;

class C { public: virtual ~C() {} };

main() {
  ifstream f("pippo"); fstream g("pluto"), h("zagor"); iostream* p = &h;
  C c1,c2;
  cout << Fun(&cout,cin) << endl; // stampa: 0
  cout << Fun(&cout,cerr) << endl; // stampa: 1
  cout << Fun(p,h) << endl; // stampa: 0
  cout << Fun(&f,*p) << endl; // stampa: 0
  cout << Fun(&g,h) << endl; // stampa: 1
  cout << Fun(&c1,c2) << endl; // stampa: 0
}
```

4. Si consideri la seguente gerarchia di classi e le seguenti variabili globali:

```
class A {public: virtual ~A() {} };
class B: virtual public A {};
class C: virtual public A {};
class D: virtual public A {};
class E: virtual public C {};
class F: virtual public C {};
class G: public B, public E, public F {};

B b; D d; E e; F f; G g; A* pa; B* pb; C* pc; F* pf;
```

   Per ciascuno dei seguenti `main()` segnare con una croce (NON COMPILA) per indicare la mancata compilazione mentre in caso di corretta compilazione scrivere nell'apposito spazio riservato dai puntini la stampa prodotta (`OK` oppure `NO`) in output dall'esecuzione.

```
main() {pc = &e; cout << (dynamic_cast<D*> (pc) ? "OK" : "NO");}   (NON COMPILA)  .....
main() {cout << (dynamic_cast<B*> (&g) ? "OK" : "NO");}            (NON COMPILA)  .....
main() {pa = &f; cout << (dynamic_cast<C*> (pa) ? "OK" : "NO");}   (NON COMPILA)  .....
main() {pb = &b; cout << (dynamic_cast<G*> (pb) ? "OK" : "NO");}   (NON COMPILA)  .....
main() {cout << (dynamic_cast<D*> (&d) ? "OK" : "NO");}            (NON COMPILA)  .....
main() {pf = &g; cout << (dynamic_cast<E*> (pf) ? "OK" : "NO");}   (NON COMPILA)  .....
main() {pf = &f; cout << (dynamic_cast<E*> (pf) ? "OK" : "NO");}   (NON COMPILA)  .....
```

5.
```
#include<iostream>
using namespace std;

class A {
protected:
  int k;
  A(int x=1): k(x) {}
  virtual ~A() {}
public:
  virtual void m() {cout << k << " A::m() ";}
  virtual void m(int x) {k=x; cout << k << " A::m(int) ";}
};
class B: virtual public A {
public:
  virtual void m(double y) {cout << k << " B::m(double) ";}
  virtual void m(int x) {cout << k << " B::m(int) ";}
};
class C: virtual public A {
public:
  C(int x = 2): A(x) {}
  virtual void m(int x) {cout << "C::m(int) ";}
};
class D: public B {
public:
  D(int x=3): A(x) {}
  virtual void m(double y) {cout << "D::m(double) ";}
  virtual void m() {cout << "D::m() ";}
};
class E: public D, public C {
public:
  E(int x=4): C(x) {}
  virtual void m() {cout << "E::m() ";}
  virtual void m(int x) {cout << "E::m(int) ";}
  virtual void m(double y) {cout << "E::m(double) ";}
};
main() {
  A* a[7] = {new B(),new C(),new D(),new E(),new C(5),new D(6),new E(7)};
  for(int i=0; i<7; ++i) {
    a[i]->m(); a[i]->m(i); cout << " *** " << i << endl;
  }
}
```

Questo programma compila correttamente. Quali stampe produce la sua esecuzione? Se una istruzione non produce alcuna stampa (oltre a quella già indicata) si scriva **NESSUNA STAMPA**.

..... *** 0

..... *** 1

..... *** 2

..... *** 3

..... *** 4

..... *** 5

..... *** 6