Programmazione 2
Appello d'Esame – 11/12/2006

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

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

Non si possono consultare appunti e libri. Dove previsto scrivere CHIARAMENTE la risposta nell'apposito spazio. ATTENZIONE: In tutti gli esercizi si intende la compilazione standard `g++` con il flag `-fno-elide-constructors`.

Esercizio 1

Si consideri la seguente realtà. Il provider internet SlowWeb© offre abbonamenti ADSL con tariffazione a tempo oppure a traffico.

1. Definire una classe `Abbonato` i cui oggetti rappresentano un abbonato ADSL a SlowWeb. La classe `Abbonato` dichiara un metodo virtuale puro `double costoAttuale()` che prevede il seguente contratto: una invocazione `a.costoAttuale()` ritorna il costo attualmente da pagare nel mese corrente per l'abbonato `a`.

2. Definire una classe `AbbonatoTempo` derivata da `Abbonato` i cui oggetti rappresentano un abbonato a SlowWeb con tariffazione a quantità di tempo di connessione. Un `AbbonatoTempo` è caratterizzato dal totale dei secondi di connessione nel mese corrente. Per tutti gli abbonati con tariffazione a tempo il costo per secondo di connessione è fissato in 0.2 eurocent. La classe `AbbonatoTempo` implementa quindi `costoAttuale()` ritornando il costo attualmente da pagare nel mese corrente per un dato abbonato con tariffazione a tempo.

3. Definire una classe `AbbonatoTraffico` derivata da `Abbonato` i cui oggetti rappresentano un abbonato a SlowWeb con tariffazione a quantità di traffico effettuato. Un `AbbonatoTraffico` è caratterizzato dal totale di KB di traffico effettuato nel mese corrente. Per tutti gli abbonati con tariffazione a traffico il costo per KB di connessione è fissato in 0.1 eurocent. La classe `AbbonatoTraffico` implementa quindi `costoAttuale()` ritornando il costo attualmente da pagare nel mese corrente per un dato abbonato con tariffazione a traffico.

4. Definire una classe `FilialeSlowWeb` i cui oggetti rappresentano un insieme di abbonati gestiti da una filiale di SlowWeb. Una `FilialeSlowWeb` pratica degli sconti agli abbonati che effettuano elevate quantità di connessioni. Quindi una `FilialeSlowWeb` è anche caratterizzata da un importo di sconto Sc e da una soglia di secondi S e da una soglia di KB K oltre cui pratica all'abbonato lo sconto Sc. Devono essere disponibili nella classe `FilialeSlowWeb` le seguenti funzionalità:

   - Un metodo `void inserisci(const Abbonato&)` con il seguente comportamento: una chiamata `fil.inserisci(a)` aggiunge l'abbonato `a` all'insieme di abbonati gestiti dalla filiale `fil`.
   - Un metodo `double bolletta(const Abbonato& a)` con il seguente comportamento: una chiamata `fil.bolletta(a)` ritorna la bolletta attualmente da pagare per il mese corrente dall'abbonato `a` alla filiale `fil`: se il tempo di connessione dell'abbonato a tempo `a` supera la soglia S allora dal totale da pagare viene detratto lo sconto Sc; se la quantità di traffico effettuato dall'abbonato a traffico `a` supera la soglia K allora dal totale da pagare viene detratto lo sconto Sc.
   - Un metodo `double totaleBollette()` con il seguente comportamento: una chiamata `fil.totaleBollette()` ritorna l'importo totale delle bollette attualmente da pagare da tutti gli abbonati gestiti dalla filiale SlowWeb `fil`.

Esercizio 2

```
class A {
public: A() {cout << "A ";}
};

class B: virtual public A {
public: B() {cout << "B ";}
};

class C: virtual public A {
public: C(): A() {cout << "C ";}
};

class D: virtual public B, virtual public C {
public: D(): C(), B() {cout << "D ";}
};
```

Le precedenti classi compilano correttamente. Si supponga che le precedenti classi siano visibili alle seguenti definizioni.

```
class E: public D {
  public: E(): B() {cout << "E ";}
};
class F: virtual public E {
  public: F() {cout << "F ";}
};
```
risposta: .....

```
class E: public D {
  public: E(): B() {cout << "E ";}
};
class F: public E {
  public: F() {cout << "F ";}
};
```
risposta: .....

```
class F: public B, virtual public C {
  public: F()  {cout << "F ";}
};
```
risposta: .....

```
class E: public B {
  public: E() {cout << "E ";}
};
class F: public E,  virtual public C {
  public: F() {cout << "F ";}
};
```
risposta: .....

```
class E: virtual public B {
  public: E() {cout << "E ";}
};
class F: public E,  virtual public C {
  public: F() {cout << "F ";}
};
```
risposta: .....

```
class E: public B {
  public: E() {cout << "E ";}
};
class F: virtual public E,  virtual public C {
  public: F() {cout << "F ";}
};
```
risposta: .....

Per ognuna delle precedenti definizioni della classe `F` scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione della classe `F` provoca un errore;
- se invece la classe `F` compila correttamente, la stampa prodotta in output dall'esecuzione di:

  ```
  main() { F f; }
  ```

Esercizio 3

Siano `A`, `B`, `C` e `D` classi polimorfe con definizioni visibili al seguente frammento di codice.

```
A& fun(B& ref) {return ref;}

main() {
  B b;
  B& refb = b;
  C c;
  fun(refb) = c;
  D d;
  B& ref = d;
  if(dynamic_cast<B*>(new C())) cout << "UNO ";
  else cout << "DUE ";
  if(!dynamic_cast<C*>(&fun(b))) cout << "TRE ";
}
```

Si supponga che:

1. il precedente `main()` compili correttamente ed esegua senza provocare errori a run-time;
2. l'esecuzione del `main()` provochi su `cout` la stampa di `DUE TRE`.

Disegnare nello spazio sottostante i diagrammi di **tutte** le possibili gerarchie di tipo per le classi `A`, `B`, `C` e `D` compatibili con le precedenti ipotesi.

Esercizio 4

Si considerino le seguenti definizioni, la cui compilazione non provoca errori.

```
class A {
public:
  virtual void m() {cout << " A::m() ";}
  virtual void m(int x) {cout << " A::m(int) ";}
};
class B: public A {
public:
  virtual void m(bool b) {m(3); cout << " B::m(bool) ";}
  void m(int x) {cout << " B::m(int) ";}
};
class C: public A {
public:
  virtual void m(bool b) {cout << " C::m(bool) ";}
};
class D: public B {
public:
  void m() {B::m(false); cout << " D::m() ";}
  void m(int x) {cout << " D::m(int) ";}
};
class E: public D {
public:
  void m(int x) {cout << " E::m(int) ";}
};
A a; B b; C c; D d; E e;
```

Si supponga che ognuno dei seguenti frammenti sia il codice di un `main()` che può accedere alle precedenti definizioni. Si scriva nell'apposito spazio contiguo:

- **NON COMPILA** quando tale `main()` non compila;
- **ERRORE RUN-TIME** quando tale `main()` compila ma la sua esecuzione provoca un errore a run-time;
- la stampa che produce in output su `cout` nel caso in cui tale `main()` compili ed esegua senza errori; se non provoca alcuna stampa si scriva **NESSUNA STAMPA**.

`B* pb=&c; pb->m(3);`	
`B* pb=&e; pb->m(true);`	
`B* pb=&e; pb->m(3);`	
`A* pa=&e; pa->m();`	
`A* pa=&d; pa->m();`	
`D* pd=&e; pd->m(true);`	
`B* pb=&e;` `D* pd=dynamic_cast<D*>(pb); pd->m(true);`	
`A* pa=&b; pa->m(false);`	
`D* pd=&e; pd->m(4);`	
`B* pb=&d; pb->m(true);`	
`B* pb=&d; pb->m(5);`	