IV Appello Programmazione ad Oggetti – 07/07/2016

Esercizio 1

Si consideri il seguente modello concernente una rubrica dei contatti di uno smartphone.

(A) Definire una classe *Nome* i cui oggetti rappresentano il nome attribuibile ad un contatto di una rubrica di uno smartphone. Si suppone che ogni oggetto *Nome* è semplicemente caratterizzato da una stringa che identifica il nome. Definire una classe *Numero* i cui oggetti rappresentano il numero di telefono associato ad un contatto telefonico di una rubrica di uno smartphone. Si suppone che ogni oggetto *Numero* è rappresentato semplicemente da una stringa.

(B) Definire la seguente gerarchia di classi.

1. Definire una classe base polimorfa astratta *Contatto* i cui oggetti rappresentano un contatto di una rubrica di uno smartphone. Ogni *Contatto* è caratterizzato dal nome associato. La classe è astratta in quanto prevede i due seguenti metodi virtuali puri:
   - un metodo di clonazione polimorfa;
   - l'overloading dell'operatore di uguaglianza: *bool operator==(const Contatto&)*.
2. Definire una classe concreta *Telefonico* derivata da *Contatto* i cui oggetti rappresentano un contatto telefonico di una rubrica di uno smartphone. Ogni oggetto *Telefonico* è caratterizzato dal *Numero* di telefono, dal numero di SMS inviati a quel numero e dall'essere disponibile anche via Whatsapp o no, La classe Telefonico implementa i metodi virtuali puri di Contatto come segue:
   - implementazione standard del metodo di clonazione polimorfa
   - per ogni *t* di tipo *Telefonico* e per ogni puntatore *q* a *Contatto*, *t == *q* ritorna true se e soltanto se *q è di tipo (o sottotipo di) *Telefonico* e i due oggetti *t* e **q* hanno lo stesso numero di telefono; in tutti gli altri casi ritorna false.
3. Definire una classe concreta *Skype* derivata da *Contatto* i cui oggetti rappresentano un contatto skype di una rubrica di uno smartphone. Ogni oggetto Skype è caratterizzato da uno username (che si assume essere semplicemente una stringa) e dall'essere disponibile a videochiamate Skype oppure no. La classe Skype implementa i metodi virtuali puri di *Contatto* come segue:
   - implementazione standard del metodo di clonazione polimorfa
   - per ogni *s* di tipo *Skype* e per ogni puntatore *q* a *Contatto*, *s == *q* ritorna true se e soltanto se **q* è di tipo (o sottotipo di) *Skype*, e i due oggetti *s* e **q* hanno lo stesso username, in tutti gli altri casi ritorna false.

(C) Definire una classe *Rubrica* i cui oggetti rappresentano la rubrica dei contatti di uno smartphone. La classe *Rubrica* deve soddisfare le seguenti specifiche:

1. è definita una classe annidata *Entry* i cui oggetti rappresentano un contatto inserito nella rubrica. Ogni oggetto *Entry* è rappresentato da un puntatore polimorfo ad un *Contatto* e dal numero totale di comunicazioni (di qualsiasi natura) con quel contatto.
   - La classe *Entry* deve essere dotata di un opportuno costruttore *Entry(Contatto*, int)* con il seguente comportamento: *Entry(p, x)* costruisce un oggetto *Entry* il cui puntatore polimorfo punta ad una copia dell'oggetto **p* ed il cui numero totale di comunicazioni con **p* è uguale ad *x*.
   - La classe *Entry* ridefinisce costruttore di copia profonda, essegnazione profonda e distruttore profondo.
2. Un oggetto di *Rubrica* è caratterizzato da un vector di oggetti *Entry* contenente tutti i contatti della rubrica
3. La classe *Rubrica* rende disponibili i seguenti metodi:
   - Un metodo *void insert(Contatto*, int)* con il seguente comportamento: una invocazione *rub.insert(p, n)* inserisce il nuovo oggetto *Entry(p, n)* nel vector dei contatti della rubrica *rub* e il contatto **p* non è già presente in *rub*, mentre se il contatto **p* risulta già presente incrementa il numero totale di comunicazioni con **p* della quantità *n*.
   - Un metodo *list<Telefonico> tel(const Nome&, int)* con il seguente comportamento: una invocazione *rub.tel(name, s)* ritorna uan lista di oggetti *Telefonico* contenente tutti e soli i contatti telefonici presenti nella rubrica *rub* che: hanno *name* come nome e ed il cui numero di SMS a loro inviati è >= *s*.

- Un metodo *int whatsCall()* con il seguente comportamento: una invocazione *rub.whatsCall()* ritorna il numero di contatti presenti nella rubrica *rub* che: o sono contatti telefonici disponibili via WhatsApp oppure sono contatti Skype disponibili a video chiamate.

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 (con opportuni *include* e *using*). Si supponga che le precedenti classi siano visibili ai seguenti frammenti di codice.
Per ognuno scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione del codice provoca un errore;
- **ERRORE RUN-TIME** se il codice compila correttamente ma l-esecuzione di *main() {F f;}* provoca un errore run-time;
- se invece il codice compila correttamente e l'esecuzione di *main() {F f;}* non provoca errori run-time, la stampa prodotta in output su *cout*; se non provoca nessuna stampa scrivere **NESSUNA STAMPA**.

```
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: ….....

```
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: ….....