Programmazione 2
Appello d'esame – 6/12/2002

Nome..... Cognome.....
Matricola..... Laurea in.....

Tempo a disposizione: 2H45M**. Non si possono consultare appunti e libri. Ogni quesito a risposta multipla ha ESATTAMENTE una risposta corretta: segnare con una croce la risposta scelta. Se una risposta corretta vale 1, allora una risposta errata viene penalizzata con -0.5, mentre nessuna risposta vale 0. Dove previsto scrivere CHIARAMENTE la risposta nell'apposito spazio.**

1. Si considerino le seguenti definizioni di classe e funzione:

```
#include<typeinfo>

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

char F(A* p, C& r) {
  B* punt1 = dynamic_cast<B*> (p);
  try{
    E& s = dynamic_cast<E&> (r);
  }
  catch(bad_cast) {
    if(punt1) return 'O';
    else return 'M';
  }
  if(punt1) return 'R';
  return 'A';
};
```

Si consideri inoltre il seguente `main()` incompleto, dove `?` è semplicemente un simbolo per una incognita:

```
#include<iostream>
main(){
  A a; B b; C c; D d; E e;
  cout << F(?,?) << F(?,?) << F(?,?) << F(?,?);
  }
```

Definire opportunamente le chiamate in tale `main()` (usando gli oggetti locali `a`, `b`, `c`, `d`, `e`) in modo tale che la sua esecuzione provochi la stampa: `ROMA`.

```
main(){
  A a; B b; C c; D d; E e;

  cout << F(..........,..........) <<

          F(..........,..........) <<

          F(..........,..........) <<

          F(..........,..........);
}
```

2. Il seguente programma compila.

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

class Z {
public:
  Z(int x=9) {cout << x << " Z01 ";};
};

class A {
protected:
  int i;
public:
  A() {cout << "A() ";};
  A(int x): i(x) {cout << "A(int) ";};
  virtual ~A() {};
};

class B: public A {
public:
  B(): A(3) {cout << "B() ";};
  B(int y) {cout <<"B(int) ";};
};

class C: virtual public B {
protected:
  A campoDati;
};

class D: virtual public B {
private:
  Z z;
public:
  D(): z(7) {cout << "D() ";};
  D(char c) {cout << "D(char) ";}
};

template<class T=Z>
class E: public C, public D {
private:
  T t;
public:
  E(): t(4), B(3), D('@') {cout << "E() ";}
};

class F: public E<int> {
public:
  B campoDati;
  F(B x): campoDati(x) {cout << "F() ";}
};

main() {
  A* p = new C(); cout << "UNO\n";
  B* q = dynamic_cast<B*>(p); cout << "DUE\n";
  E<> e; cout << "TRE\n";
  A a[3]={D(), B(3)}; cout << "QUATTRO\n";
}
```

Cosa stampa in output?

..... UNO

..... DUE

..... TRE

..... QUATTRO

3. Si consideri ancora la gerarchia di classi dell'esercizio 2 e si consideri il seguente:

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

Cosa stampa in output?

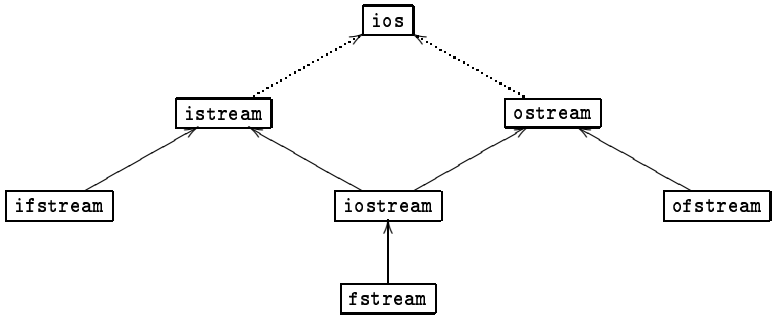
(a) `A() B() A(int) B() A() 9 Z01 D(char) E() F()`

(b) `A() B() A() B() A() 9 Z01 D(char) E() F()`

(c) `A(int) B() A(int) B() A() 9 Z01 D(char) E() F()`

(d) `A(int) B() A() B() A() 9 Z01 D(char) E() F()`

4. Si consideri la gerarchia di classi per l'I/O riportata in figura (l'intera gerarchia è visibile tramite la direttiva `#include<fstream>`)



La classe base `ios` ha il distruttore virtuale, il costruttore di copia privato ed un unico costruttore (a 2 parametri con valori di default) protetto. Diciamo che le classi derivate da `istream` ma non da `ostream` (ad esempio `ifstream`), e `istream` stessa, sono *classi di input*, le classi derivate da `ostream` ma non da `istream` (ad esempio `ofstream`), ed `ostream` stessa, sono *classi di output*, mentre le classi derivate sia da `istream` che da `ostream` sono classi di I/O (esempi: `iostream` e `fstream`). Quindi ogni classe di input, output o I/O è una sottoclasse di `ios`. Definire una funzione `int F(ios& ref)` che restituisce -1 se il tipo dinamico di `ref` è un riferimento ad una classe di input, 1 se il tipo dinamico di `ref` è un riferimento ad una classe di output, 0 se il tipo dinamico di `ref` è un riferimento ad una classe di I/O, mentre in tutti gli altri casi lancia un'opportuna eccezione.

Quindi, ad esempio, il seguente `main()` provoca la stampa riportata.

```
main() {
  istream b; ostream c; iostream d; ifstream e; ofstream f; fstream g;
  cout << F(b) <<' '<< F(c) <<' '<< F(d) <<' '<< F(e) <<' '<< F(f) <<' '<< F(g);
}
// stampa: -1 1 0 -1 1 0
```

5. Si completi la frase: "Se il distruttore di una classe base `B` è dichiarato virtuale ..." in modo da ottenere un'affermazione vera:

(a) "... allora i distruttori delle sottoclassi di `B` derivate virtualmente non sono virtuali"

(b) "... allora i distruttori di tutte le sottoclassi di `B` diventano automaticamente virtuali"

(c) "... allora anche ogni costruttore di `B` deve essere dichiarato virtuale"

(d) "... allora è necessario ridefinire il distruttore in tutte le sottoclassi di `B`"

6. Quale delle seguenti affermazioni è falsa?

(a) "È possibile derivare una sottoclasse non template da una istanza di una classe base template"

(b) "È possibile derivare una sottoclasse non template da una classe base template"

(c) "È possibile derivare una sottoclasse template da una classe base template"

(d) "È possibile derivare una sottoclasse template da una classe base non template"

7. Definire una superclasse `Studente` e due sue sottoclassi `StudenteIC` e `StudenteFC` che formano una gerarchia di classi i cui oggetti rappresentano studenti di una certa Università, distinti tra studenti in corso (`StudenteIC`) e studenti fuori corso (`StudenteFC`). Ci interesserà rappresentare delle informazioni utili per il calcolo delle tasse universitarie. La gerarchia deve soddisfare le seguenti specifiche:

   - Un oggetto `Studente` è caratterizzato dal nome, dal corso di laurea frequentato, dalla durata legale in anni del corso di laurea (quindi ≥ 3 e ≤ 6), dal numero totale di esami previsti dal corso di laurea (diciamo ≥ 15 e ≤ 60), dal numero di esami sostenuti (quindi non negativo e minore o uguale al numero totale di esami previsti), dal voto medio degli esami sostenuti. Tutte queste informazioni devono essere private. La classe non deve essere astratta, ma comunque deve essere progettata in modo tale che in ogni funzione esterna ed in ogni classe non derivata da essa, non sia possibile costruire oggetti di `Studente`.
   - Un oggetto della sottoclasse `StudenteIC` è caratterizzato dall'anno di corso, che deve quindi essere compreso tra 1 e la durata legale in anni del corso di laurea, e dal reddito annuale del proprio nucleo familiare. Queste informazioni devono essere private. È definito un metodo pubblico `int classeDiReddito()` che ritorna la classe di reddito di uno studente in corso: classe 0 se il reddito annuale è ≤ 15000 euro e lo studente frequenta l'ultimo anno di corso, classe 1 se il reddito annuale è ≤ 15000 euro ma lo studente non frequenta l'ultimo anno di corso, classe 2 se il reddito annuale è > 15000 e ≤ 30000 euro, classe 3 se il reddito annuale è > 30000 euro.
   - Un oggetto della sottoclasse `StudenteFC` è caratterizzato dal numero di anni di fuori corso, ovvero un intero ≥ 1. Tale informazione deve essere privata. È definito un metodo pubblico `bool bonus()` che determina se lo studente fuori corso ha diritto ad un bonus nella tassazione: il metodo ritorna `true` se e soltanto se il numero di esami ancora da sostenere è < 5.

   Si chiede inoltre di definire esternamente alla gerarchia (quindi senza alcuna relazione di ereditarietà con classi della gerarchia) una classe `Tasse` da usarsi per determinare la tassa di iscrizione annuale per un qualsiasi studente. La classe deve rappresentare le seguenti informazioni: (1) l'importo base di tassazione annuale, (2) l'importo della penale di tassazione e (3) l'importo del bonus di tassazione. Tali informazioni non devono essere pubbliche. **La classe `Tasse` non deve essere dichiarata friend in nessun'altra classe.** La classe `Tasse` contiene un metodo pubblico statico `int calcolaTasse(Studente& s)` che calcola la tassa di iscrizione annuale dovuta dallo studente `s` nel seguente modo:

   – se `s` è uno studente in corso, allora la tassa dovuta è data dall'importo base di tassazione annuale sommato con l'importo della penale di tassazione moltiplicato per la classe di reddito dello studente, e da tale somma si detrae l'importo del bonus di tassazione qualora il voto medio degli esami sostenuti è ≥ 28.

   – se `s` è uno studente fuori corso, allora la tassa dovuta è data dall'importo base di tassazione annuale sommato con il triplo dell'importo della penale di tassazione moltiplicato per il numero di anni di fuori corso, e da tale somma si detrae l'importo del bonus di tassazione quando lo studente ha diritto al bonus.

   Definire infine un esempio di metodo `main()` che invoca esattamente tre volte il metodo `calcolaTasse()` di `Tasse` producendo precisamente il seguente output:

```
Lo studente fuori corso di Matematica Pippo deve pagare 2000 euro di tasse.
Lo studente in corso di Informatica Pluto deve pagare 1600 euro di tasse.
Lo studente fuori corso di Fisica Paperino deve pagare 1800 euro di tasse.
```