Programmazione 2
Seconda Prova Parziale – 28.11.2002

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

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

Tempo a disposizione: 2H30M**. 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<iostream>
#include<string>
#include<typeinfo>

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

class B: virtual public A { };

class C: virtual public A { };

class D: public B, public C { };

int F(A& r, A* p) {
  if(typeid(r)==typeid(*p)) return 1;
  D* q = dynamic_cast<D*> (p);
  if(q) return 2;
  try{
    B& s = dynamic_cast<B&> (r); return 3;
  }
  catch(...) {return 4;}
};
```

   Quale dei seguenti `main()` provoca la stampa `1234`?

   (a)
```
main(){ A a; B b; C c; D d;
     cout << F(a,&a) << F(a,&d) << F(c,&b) << F(a,&c); }
```
   (b)
```
main(){ A a; B b; C c; D d; C* q=&d;
     cout << F(a,&a) << F(d,q) << F(d,&b) << F(a,&c); }
```
   (c)
```
main(){ A a; B b; C c; D d;
     cout << F(a,&a) << F(a,&d) << F(d,&b) << F(b,&c); }
```
   (d)
```
main(){ A a; B b; C c; D d; C* q=&d;
     cout << F(d,&d) << F(a,q) << F(d,&b) << F(a,&c); }
```

2. Si completi la frase: "Generalmente con il termine smart pointer ci si riferisce a ..." in modo da ottenere un'affermazione vera:

   (a) "... un oggetto di una speciale classe con un unico campo dati di tipo puntatore"

   (b) "... un puntatore ad un tipo qualsiasi"

   (c) "... un puntatore generico di tipo `void*` con comportamenti standard"

   (d) "... un puntatore generico di tipo `void*` per cui sono stati ridefiniti assegnazione, costruttore di copia e distruttore"

3. Il seguente programma compila.

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

class B {
public:
  string r;
  B(string x="pippo "): r(x) {};
  virtual void F(B* p) {cout << r + p->r;};
  virtual void G() {cout << r;};
};

class D: public B {
private:
  string s;
public:
  D(string x="pluto "): s(x) {};
  virtual void F(B* p) {
    D* q = dynamic_cast<D*> (p);
    if(q) {cout << s + q->s;}
    else {cout << s + p->r;}
  };
  virtual void G() {cout << s;};
};

class E: public B {
private:
  string t;
public:
  E(string x="topolino "): t(x) {};
  virtual void F(B* p) { cout << r + t + p->r;};
  virtual void G() {cout << t;};
};

void H(B* p, B* q) {
  cout << "F: "; p->F(q);
  cout << "G: "; p->G();
};

main(){
  B b; D d("paperino "); E e;
  H(&b,&b); cout << "UNO\n";
  H(&e,&e); cout << "DUE\n";
  B* p = &d;
  H(p,&e); cout << "TRE\n";
  H(p,p); cout << "QUATTRO\n";
}
```

Cosa stampa in output?

..... UNO

..... DUE

..... TRE

..... QUATTRO

4. Il seguente programma compila.

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

class Z {
public:
  Z() {cout << "Z() ";};
  Z(char c) {cout << "Z(char) ";};
  Z(const Z& x) {cout << "Zc ";};
};
class A {
protected:
  int i;
public:
  A() {cout << "A() ";};
  A(int x): i(x) {cout << "A(int) ";};
};
class B: virtual public A {
public:
  B() {cout << "B() ";};
};
class C: virtual public A {
public:
  C() {cout << "C() ";};
  C(int k): A(k) {cout << "C(int) ";};
  C(string st) {cout << "C(string) ";};
};
class D: virtual public A {
private:
  Z z;
public:
  D(): z('$') {cout << "D() ";};
};

class E: public B {
public:
  static int s;
  E(int x=1) {s=s+x; cout << s << " E() ";};
};
int E::s=0;

template<class T=string>
class F: public E, public C, public D {
private:
  T t;
public:
  F(): A(3), C("pippo") {cout << "F() ";};
  F(T x): E(2), C(6) {cout << "F(" << typeid(T).name() << ") ";}
};

main() {
  C c(3); cout << "UNO\n";
  E e; cout <<"DUE\n";
  B* p = new B[2]; cout << "TRE\n";
  F<> f1; cout << "QUATTRO\n";
}
```

Cosa stampa in output?

..... UNO

..... DUE

..... TRE

..... QUATTRO

5. Si consideri ancora la gerarchia di classi dell'esercizio 4 e si consideri il seguente:

```
main() {
  Z z('*'); F<Z> f2(z);
}
```

Cosa stampa in output?

(a) `Z(char) A() B() 2 E() C(int) Z(char) D() Z() F(Z)`

(b) `Z(char) Zc A() B() 2 E() C(int) Z(char) D() Z() F(Z)`

(c) `Z(char) A() B() 1 E() C(int) Z(char) D() F(Z)`

(d) `Z(char) Zc A() B() 1 E() C(int) Z(char) D() F(Z)`

6. Si consideri il seguente programma:

```
#include<iostream>

template<class T>
class B;

template<class T>
class A {
  friend class B<T>;
private:
  T x;
public:
  A(T z): x(z) {};
};

template<class T>
class B{
private:
  A< A<T> > m;
public:
  B(T z): m(A<T>(z)) {};
  void f() {cout << (m.x).x << endl;};
};

main(){ B<char> b('*'); b.f(); }
```

Quale delle seguenti affermazioni è vera?

(a) compila e la sua esecuzione non produce alcuna stampa

(b) compila e la sua esecuzione produce la stampa: `*`

(c) compila e la sua esecuzione produce la stampa: `**`

(d) non compila

7. Si completi la frase: “La specifica esplicita delle eccezioni ...” in modo da ottenere un’affermazione vera:

   (a) “... permette all’interno di una funzione `F` di lanciare un’eccezione al chiamante di `F`”

   (b) “... permette di dichiarare tutte e sole le eccezioni che una funzione può lanciare”

   (c) “... va sempre dichiarata all’interno di un blocco try/catch”

   (d) “... causa l’invocazione della funzione `terminate()` e quindi la terminazione del programma”

8. Definire una superclasse `Messaggio` e due sue sottoclassi `Warning` ed `Errore` che definiscono una gerarchia di classi per gestire i messaggi di warning e di errore che un compilatore può segnalare in fase di compilazione. La gerarchia deve soddisfare le seguenti specifiche:

   - Un oggetto `Messaggio` è caratterizzato da una stringa che memorizza un messaggio. Tale stringa non deve essere pubblica. La classe `Messaggio` contiene un metodo pubblico `void stampaMessaggio()` virtuale puro.
   - Un oggetto della sottoclasse `Warning` è caratterizzato da un livello di gravità del messaggio di avvertimento, ovvero un intero ≥ 0. La classe `Warning` dovrà gestire il numero d’ordine degli oggetti creati in modo tale che ogni oggetto `obj` di `Warning` sia caratterizzato da un intero corrispondente al numero d’ordine di `obj`: quindi il primo oggetto creato avrà numero d’ordine 1, il secondo avrà numero d’ordine 2, etc. In `Warning` è definito concretamente il metodo `void stampaMessaggio()` che per un oggetto di invocazione stampa il messaggio di warning, seguito dal livello di gravità, seguito dal numero d’ordine. Tutti i campi dati di `Warning` devono essere privati.
   - Un oggetto della sottoclasse `Errore` è un messaggio di una delle seguenti due categorie: o un messaggio di errore di tipo, oppure un messaggio di errore non di tipo. Anche la classe `Errore` dovrà gestire il numero d’ordine degli oggetti creati, separatamente per gli oggetti che rappresentano errori di tipo ed errori non di tipo: quindi il primo oggetto di `Errore` di tipo creato avrà numero d’ordine 1, il secondo oggetto di `Errore` di tipo avrà numero d’ordine 2, il primo oggetto di `Errore` non di tipo creato avrà numero d’ordine 1, etc. In `Errore` è definito concretamente il metodo `void stampaMessaggio()` che stampa il messaggio di errore, seguito dalla categoria di errore, seguito dal numero d’ordine. Tutti i campi dati di `Errore` devono essere privati.

   Si chiede inoltre di definire una classe `Gestione_Messaggi` contenente un unico metodo pubblico statico `void stampa(Messaggio* p)` che stamperà il messaggio di warning o di errore puntato dal parametro `p`.

   Infine, si definisca inoltre un esempio di metodo `main()` che invoca esattamente cinque volte il metodo `stampa()` di `Gestione_Messaggi` producendo precisamente il seguente output:

```
Errore1 di tipo # 1
Avvertimento1 di livello 5 # 1
Errore2 non di tipo # 1
Errore3 di tipo # 2
Avvertimento2 di livello 4 # 2
```