Metodologie di Programmazione

Corso di Laurea in "Informatica"

2 febbraio 2012

Note

1. Su tutti i fogli contenenti le soluzioni indicare, IN STAMPATELLO, la data dell'appello ed il proprio cognome, nome e numero di matricola
2. Leggere attentamente i quesiti e fornire risposte sintetiche: di solito, una risposta troppo verbosa sta ad indicare che si stanno fornendo particolari non richiesti.
3. Non è consentita la consultazione di alcunché.
4. L'orario di consegna scritto alla lavagna è tassativo.
5. Il testo del compito va consegnato insieme ai fogli con le soluzioni.

1. Mostrare il processo di risoluzione dell'overloading per le seguenti chiamate di funzione. Per ogni chiamata, indicare: l'insieme delle funzioni candidate; l'insieme delle funzioni utilizzabili; la migliore funzione utilizzabile (se esiste); il motivo di eventuali errori di compilazione.

```
#include <iostream>

class Base {
public:
  void f(int, double);               // funzione #1
  void f(double, int) const;         // funzione #2
  void g(double);                    // funzione #3
  void print(std::ostream&) const;   // funzione #4
};

class Derived : public Base {
public:
  using Base::f;
  void f(double, double);            // funzione #5
  void g(double) const;              // funzione #6
private:
  void print(std::ostream&);         // funzione #7
};

int main() {
  Base b;
  Derived d;
  Base* pb = &d;
  const Derived* pd = &d;

  pb->print(std::cerr);     // chiamata (a)
  pd->print(std::cout);     // chiamata (b)

  b.f('a', 0.7);            // chiamata (c)
  d.f(12.5, 1.4);           // chiamata (d)
  pb->f(2, 0);              // chiamata (e)
  pd->f(7.2, 7);            // chiamata (f)

  const Base* pb2 = static_cast<const Base*>(pd);
  pb2->g(0.0);              // chiamata (g)
  pd->g(0.0);               // chiamata (h)
}
```

2. Si forniscano il prototipo e l'implementazione della funzione generica `count_if`, che dati in ingresso una sequenza ed un predicato unario, restituisce il numero di elementi di quella sequenza per i quali quel predicato è soddisfatto.

   Utilizzando la funzione suddetta, scrivere un programma che legge dallo standard input una sequenza di `std::string` e scrive sullo standard output il numero di stringhe lette aventi una lunghezza maggiore di 10.

3. Indicare l'output prodotto dal seguente programma.

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

struct A {
  virtual void f(int)    { cout << "A::f(int)" << endl; }
  virtual void f(double) { cout << "A::f(double)" << endl; }

  virtual void g() { cout << "A::g(double)" << endl; }

  virtual ~A() { cout << "Destructor A::~A()" << endl; }
};

struct B : public A {
  void f(int) { cout << "B::f(int)" << endl; }
  virtual void f(double) const { cout << "B::f(double) const" << endl; }

  virtual void g(int) { cout << "B::g(int)" << endl; }

  ~B() { cout << "Destructor B::~B()" << endl; }
};

struct C : public B {
  void f(int) const { cout << "C::f(int) const" << endl; }

  void g(int) { cout << "C::g(int)" << endl; }

  ~C() { cout << "Destructor C::~C()" << endl; }
};

int main() {
  A* a = new A;
  B b;
  C c;
  A& ra_b = b;
  B& rb_b = b;
  A& ra_c = c;
  B& rb_c = c;
  cout << "=== 1 ===" << endl;
  ra_b.f(1);
  rb_b.g(1);
  ra_c.f(1);
  rb_c.g(1);
  cout << "=== 2 ===" << endl;
  static_cast<A*>(&b)->f(1.2);
  static_cast<A*>(&c)->f(1);
  static_cast<B*>(&c)->g(1.2);
  cout << "=== 3 ===" << endl;
  b.f(2);
  c.g(3);
  cout << "=== 4 ===" << endl;
}
```

4. Una applicazione di rete prevede la gestione di sessioni di comunicazione di due tipologie distinte, ognuna caratterizzata da uno specifico protocollo di comunicazione. Siccome i due protocolli hanno alcune somiglianze (in termini di dati memorizzati e di operazioni di basso livello da eseguire), il programmatore ha usato il seguente progetto per incapsulare le parti comuni in una classe BasicProtocol:

```
class BasicProtocol {
private:
  /*...*/
public:
  BasicProtocol();
  virtual ~BasicProtocol();
  bool BasicMsgA( /*...*/ );
  bool BasicMsgB( /*...*/ );
  bool BasicMsgC( /*...*/ );
};

class Protocol1 : public BasicProtocol {
public:
  Protocol1();
  ~Protocol1();
  bool DoMsg1( /*...*/ );
  bool DoMsg2( /*...*/ );
  bool DoMsg3( /*...*/ );
};

class Protocol2 : public BasicProtocol {
public:
  Protocol2();
  ~Protocol2();
  bool DoMsg1( /*...*/ );
  bool DoMsg2( /*...*/ );
  bool DoMsg3( /*...*/ );
  bool DoMsg4( /*...*/ );
  bool DoMsg5( /*...*/ );
};
```

   Le funzioni membro delle classi derivate invocano le funzioni della classe base quando necessario, ma gestiscono la trasmissione indipendentemente. In particolare, gli utenti della gerarchia di classi tipicamente decidono di utilizzare uno dei due protocolli specifici. Discutere brevemente se il progetto delle classi suddette è appropriato rispetto all'uso che ne viene fatto. Elencare le eventuali modifiche da apportare, indicandone i motivi.

5. La classe seguente contiene errori inerenti la corretta gestione delle risorse. Individuare almeno due problemi logicamente distinti, indicando la sequenza di operazioni che porta alla loro occorrenza. Fornire quindi una soluzione alternativa e discutere brevemente i motivi per i quali tale soluzione si può ritenere corretta.

```
#include <string>
class A {
  int* pi;
  std::string str;
  double* pd;
public:
  A(const std::string& s) : pi(new int), str(s), pd(new double) { }
  ~A() { delete pi; delete pd; }
};
```