Metodologie di Programmazione

Corso di Laurea in "Informatica"

27 febbraio 2006

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.

```
void foo(int*);                 // Funzione #1
template <typename T>
void foo(const T&);             // Funzione #2

template <typename T>
void foo(T, T);                 // Funzione #3
template <typename T, typename U>
void foo(T, U)                  // Funzione #4
template <typename T>
void foo(const T*, T);          // Funzione #5
template <typename T, typename U>
void foo(T*, U);                // Funzione #6

namespace A {

struct Delta {
  Delta(double = 0) {};
};

template <typename T>
void bar(T, double);           // Funzione #7
template <typename T, typename U>
void bar(T, U);                // Funzione #8

} // namespace A

template <typename T>
void bar(T, const A::Delta&); // Funzione #9

int main() {
  int alfa;
  const int* p_alfa = &alfa;
  double beta;
  long gamma;
  A::Delta delta;

  foo(&alfa, beta);              // chiamata A
  foo(&alfa, alfa);              // chiamata B
  foo(p_alfa, alfa);             // chiamata C
  foo(alfa, p_alfa);             // chiamata D

  foo(alfa);                     // chiamata E
  foo(&alfa);                    // chiamata F
  foo(p_alfa);                   // chiamata G
  foo(const_cast<int*>(p_alfa)); // chiamata H

  bar(gamma, gamma);             // chiamata I
  bar(delta, alfa);              // chiamata L
  bar(delta, beta);              // chiamata M
  A::bar(beta, gamma);           // chiamata N
}
```

2. Indicare l'output prodotto dal seguente programma.

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

class Base {
public:
  Base() { cout << "Costruttore Base" << endl; }
  virtual void foo(int) { cout << "Base::foo(int)" << endl; }
  virtual void bar(int) { cout << "Base::bar(int)" << endl; }
  virtual void bar(double) { cout << "Base::bar(double)" << endl; }
  virtual ~Base() { cout << "Distruttore Base" << endl; }
};

class Derived : public Base {
public:
  Derived() { cout << "Costruttore Derived" << endl; }
  void foo(int) { cout << "Derived::foo(int)" << endl; }
  void bar(int) const { cout << "Derived::bar(int)" << endl; }
  void bar(double) const { cout << "Derived::bar(double) const" << endl; }
  ~Derived() { cout << "Distruttore Derived" << endl; }
};

void g(Base b) {
  b.foo(5);
  b.bar(5.5);
}

int main() {
  Derived derived;
  Base base;
  Base& base_ref = base;
  Base* base_ptr = &derived;
  Derived* derived_ptr = &derived;
  cout << "=== 1 ===" << endl;
  base_ptr->foo(12.0);
  base_ref.foo(7);
  base_ptr->bar(1.0);
  derived_ptr->bar(1.0);
  derived.bar(2);
  cout << "=== 2 ===" << endl;
  base.bar(1);
  derived.bar(-1.0);
  derived.foo(0.3);
  base_ptr->bar('\n');
  cout << "=== 3 ===" << endl;
  g(*derived_ptr);
  return 0;
}
```

3. Un'applicazione software si interfaccia ad un'apparecchiatura di comunicazione multi-funzione (fax e modem) mediante il corrispondente driver proprietario, utilizzando un file header come il seguente:

```
// File FaxModem.hh
class FaxModem_AllStars {
private:
  // ...
public:
  void fax_function_1();
  void fax_function_2(const char*);
  void fax_function_3(int);
  // ...
  void modem_function_1(const char*);
  void modem_function_2();
  void modem_function_3(unsigned int);
  // ...
};
```

Il codice utente, nel quale le due funzionalità dell'apparecchiatura sono sempre utilizzate in contesti distinti, è il seguente:

```
// File User.cc
#include "FaxModem.hh"

void user_function_1(FaxModem_AllStars& f) {
  f.fax_function_1();
  // ...
  f.fax_function_3(12);
  // ...
}

void user_function_2(FaxModem_AllStars& m, const char* command) {
  m.modem_function_1(command);
  // ...
  m.modem_function_3(1024);
  // ...
}

void user_function_3(FaxModem_AllStars& f, FaxModem_AllStars& m) {
  f.fax_function_2("+390521906950");
  // ...
  m.modem_function_2();
  // ...
}
```

L'utente vuole eliminare la dipendenza del proprio codice dal produttore AllStars, in quanto sul mercato esistono altri produttori di apparecchiature multifunzione analoghe, nonché produttori di apparecchiature che supportano una sola delle due funzionalità.

Mostrare le modifiche da apportare in tal senso al codice utente, comprese eventuali nuove interfacce. In particolare, mostrare come il driver proprietario di AllStars (non modificabile dall'utente) possa essere interfacciato con il nuovo codice utente.

4. Motivare brevemente l'introduzione delle guardie contro l'inclusione ripetuta dei file di intestazione, enunciando chiaramente la regola del linguaggio che verrebbe violata nel caso di un loro mancato (o scorretto) utilizzo. Scrivere un esempio, minimale ma completo, di codice che non si comporta correttamente a causa dell'assenza di tali guardie.

5. Scrivere l'implementazione dell'algoritmo della STL `remove_copy_if`, che ha come parametri una sequenza di input ed una sequenza di output, specificate mediante iteratori, ed un predicato unario. L'algoritmo copia nella sequenza di output, mantenendone l'ordine, gli elementi della sequenza di input che *non* soddisfano il predicato; l'algoritmo restituisce l'iteratore corrispondente alla fine della sequenza di output.

   Utilizzando l'algoritmo precedente, implementare l'analogo algoritmo `remove_if`, che invece di scrivere su di una sequenza di output modifica direttamente la sequenza di input. Motivare brevemente quali siano le categorie di iteratori utilizzabili in ognuno dei due algoritmi.

6. La seguente gerarchia di classi è scorretta dal punto di vista della gestione delle risorse. Scrivere semplici esempi di codice che evidenzino due *distinti* comportamenti errati del programma, spiegando il tipo di problema ed indicando il punto in cui si viene a creare. Quindi applicare alle classi le modifiche minimali che consentano di evitare tali problemi.

```
class Base {
private:
  int* pi;
  // Assegnamento: privato e non implementato.
  Base& operator=(const Base&);

public:
  Base() : pi(new int) {}
  Base(const Base& b) : pi(new int) { *pi = *(b.pi); }
  ~Base() { delete pi; }
  int foo() { return *pi; }
};

class Derived : public Base {
private:
  int* pj;
  // Assegnamento: privato e non implementato.
  Derived& operator=(const Derived&);

public:
  Derived() : pj(new int) {}
  ~Derived() { delete pj; }
  int foo() { return *pj + Base::foo(); }
};
```