Metodologie di Programmazione

Corso di Laurea in "Informatica"

06 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.

```
#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;

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

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

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

2. Indicare l'output prodotto dal seguente programma.

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

class A {
public:
  A() { cout << "Constructor A::A()" << endl; }

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

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

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

class B {
public:
  B() { cout << "Constructor B::B()" << endl; }

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

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

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

class D : public B, public A {
public:
  D() { cout << "Constructor D::D()" << endl; }

  void f(int) { cout << "D::f(int)" << endl; }

  using A::g;
  void g(int) { cout << "D::g(int)" << endl; }

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

void h(A a, B b, D& d) {
  a.g('a');
  B* pb = &b;
  pb->f(4);
  d.g(44);
}

int main() {
  D d;
  A& ra = d;
  B& rb = d;
  cout << "=== 1 ===" << endl;
  ra.f(1);
  ra.g(1);
  rb.f(1);
  rb.g(1);
  cout << "=== 2 ===" << endl;
  d.f(1.2);
  d.g(1);
  d.g(1.2);
  cout << "=== 3 ===" << endl;
  h(d, d, d);
  cout << "=== 4 ===" << endl;
  return 0;
}
```

3. Uno strumento software utilizza codice come il seguente allo scopo di gestire la produzione automatica di documentazione secondo diversi formati di stampa.

```
#include <string>
using std::string;

class Manual_Generator {
public:
  virtual void put(const string& s) = 0;
  virtual void set_boldface() = 0;
  virtual void reset_boldface() = 0;
  // ...
};

class HTML_Generator : public Manual_Generator {
public:
  void put(const string& s);
  void set_boldface();
  void reset_boldface();
  void hyperlink(const string& uri, const string& text);
  // ...
};

class ASCII_Generator : public Manual_Generator {
public:
  void put(const string& s);
  void set_boldface();
  void reset_boldface();
  void page_break();
  // ...
};

void f(Manual_Generator* mg_p) {
  // ...
  HTML_Generator* html_p = dynamic_cast<HTML_Generator*>(mg_p);
  if (html_p)
    html_p->hyperlink("http://www.cs.unipr.it/ppl", "PPL");
  else
    mg_p->put("PPL (http://www.cs.unipr.it/ppl)");
  // ...
  ASCII_Generator* ascii_p = dynamic_cast<ASCII_Generator*>(mg_p);
  if (ascii_p)
    ascii_p->page_break();
  else
    // Nota: usare il tag HR per simulare il cambio pagina in HTML.
    mg_p->put("<HR>");
  // ...
}
```

Enunciare il principio della programmazione orientata agli oggetti che viene violato nel codice precedente e mostrare (brevemente) le conseguenze di questa violazione. Fornire una soluzione alternativa che rispetti il principio in questione, mostrando le modifiche da apportare al codice.

4. Il codice seguente aggiunge al secondo argomento una copia degli elementi di tutte le liste contenute nel primo argomento (una lista di liste), realizzando quindi una sorta di concatenazione multipla.

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

template <typename T>
void f(const list<list<T> >& ll, list<T>& l) {
  for (typename list<list<T> >::const_iterator ll_i = ll.begin(),
         ll_end = ll.end(); ll_i != ll_end; ++ll_i)
    for (typename list<T>::const_iterator i = ll_i->begin(),
           i_end = ll_i->end(); i != i_end; ++i)
      l.push_back(*i);
}
```

   Generalizzare il codice affinché, mantenendo la funzionalità descritta, possa lavorare non solo con il tipo lista, ma con combinazioni qualunque dei contenitori sequenziali della STL.

   Generalizzare ulteriormente il codice affinché accetti come argomenti, invece di contenitori sequenziali della STL, degli iteratori. Fornire un (semplice) esempio di applicazione della versione con iteratori che non possa essere codificato utilizzando la versione con contenitori.

5. Scrivere l'implementazione dell'algoritmo della STL `find_first_of`. L'algoritmo prende in input due sequenze `[first1, last1)` e `[first2, last2)`, specificate mediante iteratori, ed un predicato binario `comp`. Utilizzando il predicato come operatore di confronto, l'algoritmo cerca nella prima sequenza il primo elemento "uguale" ad un elemento qualunque della seconda sequenza. Se lo trova, restituisce l'iteratore corrispondente; altrimenti, restituisce `last1`. Si noti che le due sequenze possono avere tipi diversi. Indicare, motivando la risposta, la più ampia categoria di iteratori utilizzabile per ognuna delle due sequenze di input.

6. Discutere brevemente se, e sotto quali condizioni, le seguenti classi si comportano correttamente dal punto di vista dell'allocazione delle risorse (si noti che quella mostrata è da considerarsi l'implementazione *completa* delle classi in questione).

```
#include <vector>

template <typename T>
class A {
private:
  int i;
  T* p;
  double d;

public:
  A() : i(), p(new T), d() {}
  ~A() { delete p; };
};

template <typename T>
class B {
private:
  int i;
  T* p;
  double d;

  B(const B&);
  B& operator=(const B&);
```

```
public:
  B() : i(100), p(new T[i]), d() {}
  ~B() { delete[] p; };
};

class C : private A<std::vector<int> > {
private:
  B<double> b1;
  B<int> b2;

  C(const C&);
  C& operator=(const C&);

public:
  C() {}
};
```