Metodologie di Programmazione

Corso di Laurea in "Informatica"

14 giugno 2010

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 e, se esiste, la migliore funzione utilizzabile.

```
namespace N {
  struct C {
    C(int);       // funzione #1
    C(const C&);  // funzione #2

    void m();     // funzione #3
    void m(int);  // funzione #4
  };

  void f(double d);        // funzione #5
  void f(const C& c);      // funzione #6
  void g(int i, double d); // funzione #7
  void g(int i, int j);    // funzione #8
  void h(C* pc);           // funzione #9

} // namespace N


void f(char); // funzione 10

int h(const char* s = 0); // funzione 11
int h(const N::C* pc);    // funzione 12


int main() {
  N::C c(5);    // chiamata A

  f(5);         // chiamata B
  f(c);         // chiamata C
  N::f('a');    // chiamata D

  g(5, 3.7);    // chiamata E
  N::g(2.3, 5); // chiamata F
  N::g(5, 2.3); // chiamata G

  h(&c);        // chiamata H
  h();          // chiamata I

  m(&c);        // chiamata J
}
```

2. 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)" << 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)" << 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;
}
```

3. Una libreria per il calcolo simbolico implementa alcune delle sue funzionalità essenziali (quali la stampa e la valutazione numerica delle espressioni simboliche) mediante codice come il seguente:

```
class Expr {
public:
  enum Kind { VAR, CONST, ADD, SUB, /*...*/ };
  Kind kind;
  // ...

  void print() const {
    switch (kind) {
    case VAR:
      print(name());
      break;
    case CONST:
      print(value());
      break;
    case ADD:
      arg1().print();
      print(" + ");
      arg2().print();
      break;
    case SUB:
      arg1().print();
      print(" - ");
      arg2().print();
      break;
    /* ... */
    }
  }

  double eval(const Var_Bindings& vb) const {
    switch (kind) {
    case VAR:
      return vb[name()];
    case CONST:
      return value();
    case ADD:
      return arg1().eval(vb) + arg2().eval(vb);
    case SUB:
      return arg1().eval(vb) - arg2().eval(vb);
    /* ... */
    }
  }

  // ...
};
```

Nell'ipotesi che l'insieme dei possibili tipi di espressioni debba essere estendibile, enunciare il principio della programmazione orientata agli oggetti che viene violato e mostrare (brevemente) le conseguenze di questa violazione.

Abbozzare una soluzione alternativa che rispetti il principio in questione, mostrando le modifiche da apportare al codice suddetto. Discutere brevemente se la soluzione proposta è anche in grado di garantire l'estendibilità dell'insieme di operazioni supportate dalla classe Expr.

4. Individuare i problemi presenti nel codice seguente e mostrare modifiche minimali per prevenirli:

```
#include <string>
#include <vector>
#include <iostream>

typedef std::vector<std::string> vect;
typedef std::vector<std::string>::iterator iter;

void f(const vect& v) {
  iter i = std::find(v.begin(), v.end(), "inizio");
  iter i_end = std::find(v.begin(), v.end(), "fine");
  while (i != i_end) {
    std::cout << *i << std::endl;
    ++i;
  }
}

void g(vect& v) {
  iter i = v.begin(), i_end = v.end();
  v.insert(++i, "prima");
  v.insert(++i, "dopo");
}
```

5. Si forniscano il prototipo e l'implementazione della funzione generica `transform` i cui parametri specificano: (a) due sequenze di ingresso (non necessariamente dello stesso tipo), di cui si assume che la seconda sia lunga almeno quanto la prima; (b) una sequenza di uscita (potenzialmente di un tipo ancora diverso); (c) una funzione binaria applicabile a coppie di elementi in ingresso (il primo preso dalla prima sequenza, il secondo dalla seconda) e che restituisce un elemento assegnabile alla sequenza di uscita.

   La funzione `transform` applica la funzione binaria agli elementi in posizione corrispondente nelle due sequenze di ingresso, assegnando il risultato ad elementi consecutivi della sequenza di uscita.

   Utilizzando la funzione suddetta, scrivere una funzione che, date due liste di interi $[i_1, \ldots, i_n]$ e $[j_1, \ldots, j_n]$ scrive sullo standard output la sequenza delle medie aritmetiche (di tipo `double`) $[(i_1 + j_1)/2, \ldots, (i_n + j_n)/2]$.

6. Il codice 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 due soluzioni alternative, rispettivamente con e senza applicazione dell'idioma RAII ("*l'acquisizione di risorse è una inizializzazione*"). Infine discutere brevemente i motivi per i quali le soluzioni fornite possono ritenersi corrette.

```
struct C { ~C() {} /* ... */ };

struct D : public C { /* ... */ };

void job(const C* pc1, const C* pc2);

void foo() {
  C* pc = new D();
  job(pc, new C());
  delete pc;
}
```