Metodologie di Programmazione

Corso di Laurea in "Informatica"

1 febbraio 2005

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 NB {
  class D {};
} // namespace NB

namespace NA {
  class C {};
  void f(int i);                          // funzione #1
  void f(double d, C c = C());            // funzione #2
  void g(C c = C(), NB::D d = NB::D());   // funzione #3
  void h(C c);                            // funzione #4

  void test1() {
    f(2.0);            // chiamata #1
  }
} // namespace NA

namespace NB {
  void f(double d);                       // funzione #5
  void g(NA::C c = NA::C(), D d = D());   // funzione #6
  void h(NA::C c, D d);                   // funzione #7

  void test2(double d, NA::C c) {
    f(d);              // chiamata #2
    g(c);              // chiamata #3
    h(c);              // chiamata #4
  }
} // namespace NB

void f(NA::C c, NB::D d);                 // funzione #8

void test3(NA::C c, NB::D d) {
  f(1.0);             // chiamata #5
  g();                // chiamata #6
  g(c);               // chiamata #7
  g(c, d);            // chiamata #8
}
```

2. Indicare l'output prodotto dal seguente programma.

```
class Base {
public:
  Base() {
    std::cout << "Constructor Base::Base()" << std::endl;
  }
  virtual void f(int) {
    std::cout << "Base::f(int)" << std::endl;
  }
  virtual void f(double) {
    std::cout << "Base::f(double)" << std::endl;
  }
  virtual void g(int) {
    std::cout << "Base::g(int)" << std::endl;
  }
  virtual ~Base() {
    std::cout << "Destructor Base::~Base()" << std::endl;
  }
};

class Derived : public Base {
public:
  Derived() {
    std::cout << "Constructor Derived::Derived()" << std::endl;
  }
  void f(char c) {
    std::cout << "Derived::f(char)" << std::endl;
  }
  void g(int) {
    std::cout << "Derived::g(int)" << std::endl;
  }
  ~Derived() {
    std::cout << "Destructor Derived::~Derived()" << std::endl;
  }
};

int main() {
  Base b;
  Derived d;
  Base& rb = b;
  Base* pb = &d;
  std::cout << "=== 1 ===" << std::endl;
  b.f(1);
  rb.f('a');
  rb.g(1);
  std::cout << "=== 2 ===" << std::endl;
  d.f(1);
  d.f(1.0);
  d.g(3.3);
  std::cout << "=== 3 ===" << std::endl;
  pb->f(1.0);
  pb->f('a');
  pb->g(3.3);
  return 0;
}
```

3. Elencare le clausole della *regola della definizione unica* (ODR – One Definition Rule) e, per ognuna di esse, fornire un esempio di violazione della regola.

4. Una compagnia telefonica utilizza codice come il seguente per calcolare l'addebito delle chiamate sulle schede prepagate:

```
class Scheda_Prepagata {
public:
  enum Tipo_Scheda { PAGA_DI_PIU, COSTO_RANDOM, PAGA_LA_MAMMA };
  Tipo_Scheda tipo_scheda() const;
  void addebita_chiamata(const Chiamata& call);
  // ...
private:
  void fai_la_cosa_giusta_1(const Chiamata& call);
  void fai_la_cosa_giusta_2(const Chiamata& call);
  void fai_la_cosa_giusta_3(const Chiamata& call);
  // ...
};

void Scheda_Prepagata::addebita_chiamata(const Chiamata& call) {
  switch (tipo_scheda()) {
  case PAGA_DI_PIU:
    fai_la_cosa_giusta_1(call);
    break;
  case COSTO_RANDOM:
    fai_la_cosa_giusta_2(call);
    break;
  case PAGA_LA_MAMMA:
    fai_la_cosa_giusta_3(call);
    break;
  }
}
```

   Quale principio della programmazione orientata agli oggetti viene violato? Enunciare il principio e spiegare le conseguenze della sua violazione. Fornire una soluzione alternativa che rispetti il principio in questione e mostrare le modifiche da apportare al nuovo codice in seguito all'introduzione di un ulteriore tipo di scheda prepagata.

5. Utilizzando gli algoritmi generici forniti dalla libreria standard, implementare la funzione

   ```
   void f(const std::vector<std::string>& vs, const char c);
   ```

   che stampa, in ordine lessicografico, tutte le stringhe in `vs` che iniziano con il carattere `c`. Fornire una versione alternativa nella quale si inverte l'ordine di stampa.

6. Si fornisca un'implementazione per l'algoritmo generico

   ```
   template <typename Iter, typename T>
   void replace(Iter first, Iter last, const T& old_value, const T& new_value);
   ```

   che rimpiazza, nella sequenza individuata da `first` e `last`, ogni occorrenza di `old_value` con una copia di `new_value`. Elencare i requisiti imposti dall'implementazione sui sei parametri della funzione.

7. Sia data la seguente definizione di classe templatica (dove si assume che le funzioni dichiarate siano state implementate correttamente):

```
template <typename T>
class Container {
public:
  Container();
  Container(const Container& y);
  Container& operator=(const Container& y);
  ~Container();
private:
  // ...
};
```

   Individuare gli errori nel codice seguente, indicandone il motivo.

```
Container<int> ci1;
Container<int> ci2 = ci1;
Container<double> cd2 = ci1;
Container<int> ci3;
Container<double> cd3;
ci3 = ci1;
cd3 = ci1;
```

   Supponendo che il codice utente di cui sopra sia ritenuto valido, come occorre modificare la classe per correggere gli errori individuati?

8. Il seguente codice non ha un comportamento corretto in presenza di eccezioni (nota: si assume che le funzioni `lock` e `unlock` facciano la cosa giusta e che la funzione `unlock` non possa lanciare eccezioni). Individuare il problema, indicando la sequenza di operazioni che porta alla sua occorrenza, e correggerlo applicando un numero minimo di modifiche al codice. Fornire quindi una soluzione alternativa applicando l'idioma "*l'acquisizione di risorse è una inizializzazione*".

```
void f() {
  lock("lock1");
  try {
    lock("lock2");
    try {
      do_the_job();
      unlock("lock2");
      unlock("lock1");
    } catch (...) {
      unlock("lock2");
    }
  } catch (...) {
    unlock("lock1");
  }
}
```