Metodologie di Programmazione

Corso di Laurea in "Informatica"

20 settembre 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.

```
#include <string>

class D;

class C {
private:
  // ...
public:
  C() {}
  C(const D&) {}
  // ...
};

class D {
private:
  // ...
public:
  D() {}
  D(const C&) {}
  // ...
};


void f(double d);                       // funzione #1
void f(int i, C c = C());               // funzione #2
void f(double d, C c = C());            // funzione #3

void g(C c, D d);                       // funzione #4
void g(D d, C c = C());                 // funzione #5

void h(const char* s);                  // funzione #6
void h(const std::string& s);           // funzione #7


int main() {
  C c;
  D d;

  f('a');          // chiamata #1
  f('a', c);       // chiamata #2
  f(3.2);          // chiamata #3

  g(c, d);         // chiamata #4
  g(d, c);         // chiamata #5
  g(c);            // chiamata #6
  g(d);            // chiamata #7

  h("abra");       // chiamata #8
  h('a');          // chiamata #9
}
```

2. Indicare l'output prodotto dal seguente programma.

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

class Derived : public Base {
public:
  Derived() {
    std::cout << "Constructor Derived::Derived()" << std::endl;
  }
  Derived(const Derived&)
    : Base() {
    std::cout << "Constructor Derived::Derived(const Derived&)" << std::endl;
  }
  void f() {
    std::cout << "Derived::f()" << std::endl;
  }
  void h() {
    std::cout << "Derived::h()" << std::endl;
  }
  ~Derived() {
    std::cout << "Destructor Derived::~Derived()" << std::endl;
  }
};

int main() {
  Base b;
  Derived d;
  std::cout << "=== 0 ===" << std::endl;
  Base& rb = b;
  Base* pb = &d;
  Base b2 = *pb;
  Base* pb2 = &b2;
  std::cout << "=== 1 ===" << std::endl;
  b.f();
  rb.f();
  rb.h();
```

```
  std::cout << "=== 2 ===" << std::endl;
  d.f();
  d.g();
  d.h();
  std::cout << "=== 3 ===" << std::endl;
  pb->f();
  pb2->f();
  pb->g();
  pb->h();
  pb2->h();
  std::cout << "=== 4 ===" << std::endl;
  return 0;
}
```

3. Indicare l'output prodotto dal seguente programma.

```
#include <iostream>

class C1 {
public:
  C1() {
    std::cerr << "Constructor C1::C1()" << std::endl;
  }
  ~C1() {
    std::cerr << "Destructor C1::~C1()" << std::endl;
  }
};

class C2 {
public:
  C2() {
    std::cerr << "Constructor C2::C2()" << std::endl;
    throw 123;
  }
  ~C2() {
    std::cerr << "Destructor C2::~C2()" << std::endl;
  }
};

class C3 {
public:
  C3() {
    std::cerr << "Constructor C3::C3()" << std::endl;
  }
  ~C3() {
    std::cerr << "Destructor C3::~C3()" << std::endl;
  }
};

class D {
private:
  C1 c1;
  C3 c3_1;
  C2 c2;
  C3 c3_2;
```

```
public:
  D() : c1(), c2(), c3_1(), c3_2() {
    std::cerr << "Constructor D::D()" << std::endl;
  }
  ~D() {
    std::cerr << "Destructor D::~D()" << std::endl;
  }
};

int main() {
  try {
    D d;
  }
  catch (char c) {
    std::cerr << "char " << c << std::endl;
  }
  catch (...) {
    std::cerr << "..." << std::endl;
  }
  return 0;
}
```

4. Una ditta di sviluppo software si occupa della manutenzione di una applicazione che, per implementare alcune sue funzionalità, utilizza la libreria "BiblioSoft", fornita da terzi e per la quale non si ha a disposizione il codice sorgente. In particolare, l'applicazione accede alla classe `BSoft`, la cui interfaccia è la seguente:

```
class BSoft {
private:
  // ...
public:
  void s1();
  void s2(const BSoft& y, int n);
  void s3(int n);
  // ...
};
```

L'utilizzo che ne viene fatto all'interno dell'applicazione è tipicamente il seguente:

```
void f(BSoft& x, const BSoft& y, int n) {
  // ...
  if (n > 0)
    x.s3(n);
  else {
    x.s1();
    n = 5;
  }
  // ...
  x.s2(y, n);
  // ...
}
```

Sul mercato esiste un'altra libreria "BiblioWare" che offre utilità simili e che è preferita da alcuni degli utenti della ditta. L'interfaccia in questo caso è la seguente:

```
class BWare {
private:
  // ...
public:
  void w1();
  void w2_1(const BWare& y);
  void w2_2(int n);
  void w3(int n);
  // ... ecc.
};
```

L'invocazione dei metodi `BWare::w1()` e `BWare::w3(n)` ha lo stesso effetto dell'invocazione di `BSoft::s1()` e `BSoft::s3(n)`, rispettivamente. I metodi `BWare::w2_1(x)` e `BWare::w2_2(n)`, se invocati immediatamente uno dopo l'altro (sullo stesso oggetto), hanno lo stesso effetto dell'invocazione di `BSoft::s2(x, n)`.

Mostrare come può essere integrato/modificato il codice precedente al fine di rendere agevole l'utilizzo di una qualunque delle due librerie (Nota Bene: non è possibile modificare il codice di `BSoft` e `BWare`).

Quale principio della programmazione orientata agli oggetti era violato nella codifica originale dell'applicazione?

5. Individuare i problemi presenti nel codice seguente:

```
#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(), "cioccolato");
  iter i_end = std::find(v.begin(), v.end(), "menta");
  while (i != i_end) {
    std::cout << *i << std::endl;
    ++i;
  }
}

void g(vect& v) {
  iter i = v.begin();
  iter i_end = v.end();
  *i = "cacao";
  v.insert(i, "vaniglia");
  while (i != i_end) {
    std::cout << *i << std::endl;
    ++i;
  }
}
```

6. Il codice seguente non ha un comportamento corretto in presenza di eccezioni. Individuare il problema, indicando la sequenza di operazioni che porta alla sua occorrenza, e correggerlo applicando l'idioma "*l'acquisizione di risorse è una inizializzazione*".

```
class One {
  int i_one;
};

class Two {
  int i_one;
  int i_two;
};

class Three {
  int i_one;
  int i_two;
  int i_three;
};

class OneTwoThree {
private:
  One one;
  Two* p_two;
  Three* p_three;

  // Dichiarati privati e non implementati.
  OneTwoThree(const OneTwoThree&);
  OneTwoThree& operator=(const OneTwoThree&);

public:
  OneTwoThree(const One& o, const Two& t) {
    one = o;
    p_two = new Two(t);
    p_three = new Three();
  }

  ~OneTwoThree() {
    delete p_two;
    delete p_three;
  }

  // ...
};
```