Metodologie di Programmazione

Corso di Laurea in "Informatica"

24 agosto 2022

Note

1. Su tutti i fogli contenenti le soluzioni indicare, IN STAMPATELLO, la data dell'appello ed il proprio cognome e nome.
2. Leggere attentamente i quesiti e fornire risposte sintetiche.
3. Non è consentita la consultazione di materiale didattico.
4. Il testo del compito va consegnato insieme a tutti i fogli con le soluzioni. Marcare in maniera evidente i fogli della brutta copia.

Esercizi solo per chi NON si avvale della prova in itinere

1. (Risoluzione overloading)

   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.

```
struct Base {
  void foo(int, double);               // funzione #1
  void foo(double, int) const;         // funzione #2
  void bar(double);                    // funzione #3
  void print(std::ostream&);           // funzione #4
};

struct Derived : public Base {
  void foo(double, double);            // funzione #5
  void bar(double) const;              // funzione #6
private:
  void print(std::ostream&) const;     // funzione #7
};

int main() {
  Base bas;
  Derived der;
  const Base* pbas = &der;
  Derived* pder = &der;
  pbas->print(std::cerr);     // chiamata (a)
  pder->print(std::cout);     // chiamata (b)
  pbas->foo(2, 0);            // chiamata (c)
  pder->foo(7.2, 7);          // chiamata (d)
  pbas->bar(0.0);             // chiamata (e)
  pder->bar(0.0);             // chiamata (f)
}
```

2. (Gestione risorse)

   Il codice seguente non si comporta bene, dal punto di vista della gestione delle risorse, in presenza di eccezioni. Individuare almeno un problema, indicando la sequenza di operazioni che porta alla sua occorrenza. Fornire quindi una soluzione basata sull'idioma RAII.

```
void foo() {
  A* a1 = new A(1);
  A* a2 = new A(2);
  job1(a1, a2)
  delete a2;
  A* a3 = new A(3);
  job2(a1, a3)
  job3(a3)
  delete a3;
  delete a1;
}
```

3. (Funzione generica)

   Definire la funzione generica `contains` che prende in input due sequenze (non necessariamente dello stesso tipo) e restituisce in output un booleano. La funzione restituisce il valore `true` se e solo se ogni elemento della seconda sequenza compare, in una posizione qualunque, nella prima sequenza. Nota: si richiede di implementare la funzione senza fare ricorso ad altri algoritmi generici della libreria standard.

Esercizi per tutti

4. (Domande a risposta aperta)

   (a) Spiegare brevemente il significato e la differenza tra invariante di classe, precondizioni e postcondizioni per un metodo.

   (b) Spiegare brevemente (aiutandosi per mezzo di semplici esempi) la differenza tra i concetti di hiding, overloading e overriding per le *funzioni membro*.

   (c) Spiegare cosa si intende quando si dice che una porzione di codice è neutrale rispetto alle eccezioni.

   (d) Spiegare brevemente cosa sono e a cosa servono gli *inseritori* della libreria standard. Scrivere un semplice esempio di codice che usa un inseritore.

   (e) Spiegare brevemente i benefici che si ottengono applicando il principio ISP (Interface Segregation Principle).

5. (Risoluzione overriding)

   Indicare l'output prodotto dal seguente programma.

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

struct A {
  A() { cout << "Constructor A::A()" << endl; }
  void f(int) { cout << "A::f(int)" << endl; }
  virtual void f(double) { cout << "A::f(double)" << endl; }
  virtual void g(double) { cout << "A::g(double)" << endl; }
  virtual ~A() { cout << "Destructor A::~A()" << endl; }
};

struct B {
  B() { cout << "Constructor B::B()" << endl; }
  virtual void f(int) { cout << "B::f(int)" << endl; }
  void f(double) { cout << "B::f(double)" << endl; }
  virtual void g(int) { cout << "B::g(int)" << endl; }
  virtual ~B() { cout << "Destructor B::~B()" << endl; }
};

struct D : public B, public A {
  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);
  d.f(1.2);
  d.g(1.2);
  cout << "=== 2 ===" << endl;
  h(d, rb, d);
  return 0;
}
```