Metodologie di Programmazione

Corso di Laurea in "Informatica"

23 gennaio 2024

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).

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

struct Derived : public Base {
  void foo(double, double);            // funzione #5
  using Base::bar;
  void bar(double) const;              // funzione #6
};

int main() {
  Derived der;
  der.foo(1.2, 0);          // chiamata (a)
  der.bar(1.2);             // chiamata (b)
  der.print(std::cout);     // chiamata (c)
  const Base& bas = der;
  bas.foo(1.2, 0);          // chiamata (d)
  bas.bar(1.2);             // chiamata (e)
  bas.print(std::cout);     // chiamata (f)
}
```

2. (Conversioni implicite) Le conversioni implicite del C++ si classificano in corrispondenze esatte (E), promozioni (P), conversioni standard (S), conversioni definite dall'utente (U).

   Assumendo che le variabili `f`, `d` e `ul` abbiano tipo, rispettivamente, `float`, `double` e `unsigned long`, per ognuno dei seguenti accoppiamenti argomento/parametro indicare (se esiste) la categoria della corrisponente conversione implicita.

Indice	Argomento	Parametro formale
(1)	`"Hello"`	`const char*`
(2)	`"Hello"`	`std::string`
(3)	`f`	`int`
(4)	`(f - d)`	`double`
(5)	`ul`	`const unsigned long&`

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. Implementare la funzione usando, in maniera appropriata, l'algoritmo generico `std::find`; elencare quindi i requisiti imposti dall'implementazione sui parametri della funzione.

4. (Gestione risorse) La classe seguente contiene errori inerenti la corretta gestione delle risorse. Individuare due problemi logicamente distinti, indicando la sequenza di operazioni che porta alla loro occorrenza. Fornire quindi una versione alternativa della classe che risolva i problemi individuati, spiegando i motivi per i quali tale soluzione si può ritenere corretta.

```
class C {
  int* pi; int* pj; std::string str;
public:
  C(const std::string& s) : pi(new int), pj(new int), str(s) { }
  ~C() { delete pi; delete pj; }
};
```

5. (Domande a risposta aperta)
   (a) Spiegare brevemente in cosa consiste l'idioma RAII/RRID, motivando come mai dovrebbe essere preferito rispetto all'uso dei blocchi try/catch.
   (b) Fornire la dichiarazione di un metodo del template di classe `std::list` che non è supportato dal template di classe `std::vector`.
   (c) Che cosa si intende quando, nel contesto di un progetto basato sul polimorfismo dinamico, si parla di "costruttori virtuali"? Rispondere fornendo anche un esempio.

6. (Risoluzione overriding) Indicare l'output prodotto dal seguente programma.

```
#include <iostream>

struct Animale {
  Animale() { std::cout << "Costr. Animale" << std::endl; }
  virtual Animale* clone() const = 0;
  virtual void verso() const = 0;
  virtual ~Animale() { std::cout << "Distr. Animale" << std::endl; }
};

struct Cane : public Animale {
  Animale* clone() const { return new Cane(*this); }
  void verso() const { std::cout << "bau!" << std::endl; }
  ~Cane() { std::cout << "Distr. Cane" << std::endl; }
};

struct Barboncino : public Cane {
  Barboncino() { std::cout << "Costr. Barboncino" << std::endl; }
  Animale* clone() const { return new Barboncino(*this); }
  void verso() const { std::cout << "wof!" << std::endl; }
  ~Barboncino() { std::cout << "Distr. Barboncino" << std::endl; }
};

int main() {
  Cane* pc = new Cane();
  pc->verso();
  Cane* pb = new Barboncino();
  pb->verso();
  Animale* pa = static_cast<Animale*>(pb);
  pa->verso();
  Animale* pa2 = pa->clone();
  delete pa2;
  delete pc;
}
```