Metodologie di Programmazione
(prova in itinere)

Corso di Laurea in "Informatica"

20 aprile 2017

Note

1. Su tutti i fogli contenenti le soluzioni indicare, IN STAMPATELLO, la data della prova ed il proprio cognome, nome e numero di matricola.
2. La prova è a *libro chiuso*: non è consentita la consultazione di libri di testo, dispense, appunti o altra documentazione; non è consentito utilizzare calcolatrici, cellulari, ecc.
3. Una volta iniziata, la prova deve essere portata a termine senza interruzioni, effettuando la consegna delle soluzioni oppure ritirandosi (in altri termini, non è consentito assentarsi temporaneamente).
4. L'orario di consegna scritto alla lavagna è tassativo: le soluzioni consegnate oltre l'orario NON saranno corrette.
5. Il testo del compito va consegnato insieme a tutti i fogli; marcare in modo evidente i fogli di brutta (che vanno anch'essi consegnati).

Esercizi

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 S {
    S(int);
  };
  void foo(int i);                    // funzione #1
  void foo(double d, S s_obj);        // funzione #2
  void bar(const char* pc);           // funzione #3
} // namespace N

void foo(double d);                   // funzione #4
void bar(char c);                     // funzione #5
void bar(unsigned long ul);           // funzione #6

void test(const N::S& s_obj) {
  N::foo(4, s_obj);   // chiamata A
  foo(4.5, s_obj);    // chiamata B
  foo(4.5, 3);        // chiamata C
  N::foo(3.3);        // chiamata D
  bar(123);           // chiamata E
  bar("123");         // chiamata F
  bar('a');           // chiamata G
}
```

2. Per ogni tipologia di conversione implicita dell'elenco, fornire un esempio minimo di codice (una dichiarazione di funzione e una corrispondente chiamata, come nell'esempio mostrato per la prima riga dell'elenco) che ne causi l'utilizzo.

   Nota bene: si richiede di dichiarare opportunamente non solo i parametri delle funzioni, ma anche eventuali tipi utente e variabili usate come argomenti delle chiamate.

Tipologia conversione	Dichiarazione	Chiamata
corrispondenza perfetta	void foo1(int)	foo1(42)
lvalue → rvalue	...	...
decadimento di array		
conversione di qualificazione		
promozione intera		
promozione floating point		
conversione standard		
conversione utente		

3. La seguente classe, intesa fornire un servizio di cifratura asimmetrica, presenta alcuni problemi di interfaccia che ne rendono l'utilizzo soggetto ad errori di programmazione evitabili. Individuare i problemi ed indicare una loro possibile soluzione (modificando l'interfaccia della classe).

```
struct Crypto {
  typedef std::string Key;
  typedef std::string Testo;
  Key k_priv;
  Key k_pubbl;
  void genera_chiavi(Key&, Key&);
  bool chiavi_accoppiate(Key& priv, Key& pubbl);
  bool check_inv() { return chiavi_accoppiate(k_priv, k_pubbl); }

  Crypto() { genera_chiavi(k_priv, k_pubbl); assert(check_inv()); }
  void codifica(Testo& chiaro_in, Testo& cifrato_out);
  void decodifica(Testo& cifrato_in, Testo& chiaro_out);
  void firma_digitale(Testo& testo_in, Testo& firmato_out);
  const Key& chiave_pubblica() { return k_pubbl; }
  // ...
};
```

4. Indicare l'output prodotto dal seguente programma.

```
#include <iostream>

struct S1 {
  S1() { std::cout << "Costr. S1" << std::endl; }
  ~S1() { std::cout << "Distr. S1" << std::endl; }
};

struct S2 {
  S2() { std::cout << "Costr. S2" << std::endl; throw 0; }
  ~S2() { std::cout << "Distr. S2" << std::endl; }
};

struct S3 {
  S3() { std::cout << "Costr. S3" << std::endl; }
  ~S3() { std::cout << "Distr ~S3" << std::endl; }
};

struct All {
  S1 s1;  S3 s3_1;  S2 s2;  S3 s3_2;

  All() : s1(), s2(), s3_1(), s3_2() {
    std::cout << "Costr. All" << std::endl;
  }
  ~All() { std::cout << "Distr. All" << std::endl; }
};

int main() {
  try { All a; }
  catch (...) { std::cout << "Problema ..." << std::endl; }
  std::cout << "Tutto bene" << std::endl;
  return 0;
}
```

5. Sapendo che le funzioni `get_lock_for`, `read_data` e `write_data` segnalano eventuali situazioni di errore lanciando eccezioni e che ogni lock acquisito con la prima funzione deve essere rilasciato invocando la funzione `release_lock_for` (sullo stesso oggetto Scheda), spiegare brevemente perché il codice seguente non gestisce correttamente le risorse.

```
void foo(Scheda& s1, Scheda& s2, Buffer data1, Buffer data2) {
  get_lock_for(s1);
  read_data(s1, data1);
  release_lock_for(s1);

  get_lock_for(s2);
  read_data(s2, data2);
  get_lock_for(s1);
  write_data(s1, data2);
  write_data(s2, data1);
  release_lock_for(s1);
  release_lock_for(s2);
}
```

Fornire una soluzione basata sull'applicazione dell'idioma RAII ("*l'acquisizione di risorse è una inizializzazione*").