Metodologie di Programmazione
(prova in itinere)

Corso di Laurea in "Informatica"

22 aprile 2016

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; la migliore funzione utilizzabile (se esiste); il motivo di eventuali errori di compilazione.

```
namespace N {

struct S {
  S(const char*);           // funzione #1
  char* get();              // funzione #2
  const char* get() const;  // funzione #3
}; // struct S

void print(const S&);       // funzione #4
char* foo(int);             // funzione #5

} // namespace N

struct A {
  A(const char*);           // funzione #6
};

void print(const A&);       // funzione #7

void foo(N::S& s, const N::S& cs)  // funzione #8
{
  const char* g1 = s.get();    // chiamata A
  char* g2 = s.get();          // chiamata B
  const char* cg1 = cs.get();  // chiamata C
  char* cg2 = cs.get();        // chiamata D
}

int main() {
  N::S s("init");    // chiamata E
  foo(s, s);         // chiamata F
  foo(12345);        // chiamata G
  print("end");      // chiamata H
}
```

2. Le conversioni implicite del C++ sono distinte nelle seguenti categorie: corrispondenze esatte (E), promozioni (P), conversioni standard (S), conversioni definite dall'utente (U).

   Assumendo che le variabili `sc`, `i`, `ul`, `f`, `ai` abbiano tipo, rispettivamente, `signed char`, `int`, `unsigned long`, `float`, `int[100]`, per ognuno dei dieci accoppiamenti argomento/parametro definiti di seguito, indicare (se esiste) la categoria della corrisponente conversione implicita.

Indice	Argomento	Parametro formale
(a)	`f`	`int`
(b)	`(ul - i)`	`unsigned long`
(c)	`12345`	`const float*`
(d)	`0`	`const float*`
(e)	`"Hello"`	`std::string`
(f)	`sc`	`int`
(g)	`5.5`	`float`
(h)	`i`	`const int&`
(i)	`ai`	`int*`
(j)	`ai[5]`	`const int*`

3. La seguente classe presenta alcuni problemi che ne rendono l'utilizzo problematico. Individuare i problemi ed indicare una possibile soluzione (riscrivendo l'interfaccia).

```
struct Matrix {
  // ...
  size_type num_rows();
  size_type num_cols();
  value_type& get(size_type row, size_type col);
  Matrix& operator-();
  Matrix& operator+=(Matrix y);
  Matrix& operator+(Matrix y);
  void print(ostrem os);
  // ...
};
```

4. Si forniscano il prototipo e l'implementazione della funzione generica `max_element` i cui parametri specificano: (a) una sequenza di ingresso; (b) un predicato binario di *confronto* (tale predicato, applicato a due elementi della sequenza di ingresso, restituisce `true` se il valore del primo argomento è minore del valore del secondo argomento). La funzione `max_element` ritorna un iteratore che individua, nella sequenza di ingresso, la prima occorrenza di un elemento avente valore massimo nella sequenza stessa, comportandosi in modo canonico in caso di sequenza vuota. Indicare (motivando la scelta) la categoria di iteratori che possono essere utilizzati per specificare la sequenza in ingresso.

5. *Utilizzando la funzione dell'esercizio precedente*, scrivere un programma che legga dallo standard input una sequenza di stringhe (`std::string`) e stampi sullo standard output: (a) la prima stringa di valore massimo secondo l'ordinamento lessicografico; (b) la prima stringa di lunghezza massima (la lunghezza di una stringa si può ottenere usando il metodo `size()`).

6. Il codice seguente presenta alcuni problemi relativi alla gestione delle risorse. Evidenziare questi problemi, differenziando tra: (a) errori che si verificano in assenza di eccezioni; (b) errori che si verificano in presenza di eccezioni.

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

   Fornire una soluzione, sempre basata sull'utilizzo dei blocchi `try` ... `catch` (ovvero, non è consentito usare smart pointers o adattatori RAII), che si comporti correttamente sia in assenza che in presenza di eccezioni. Spiegare brevemente i motivi per i quali la soluzione proposta si può dire *exception safe*.