

## Esercizi solo per chi NON si avvale della prova in itinere

1. (Progettazione tipo concreto) 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(ostream os);  
    // ...  
};
```





```
// Definire la funzione generica all_of che prende in input una sequenza  
// ed un predicato unario e restituisce in output un booleano: la  
// funzione restituisce il valore true se e solo se tutti gli elementi  
// della sequenza soddisfano il predicato. Scrivere inoltre una funzione  
// che prende in input un vettore di numeri interi e, usando la funzione  
// all_of appena definita, controlla se tutti gli interi contenuti nel  
// vettore sono negativi.
```



### 3. (Conversioni implicite)

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`, `ui`, `f`, `ai` abbiano tipo, rispettivamente, `signed char`, `unsigned int`, `float`, `int[100]`, per ognuno degli accoppiamenti argomento/parametro definiti di seguito, indicare (se esiste) la categoria della corrispondente conversione implicita:

Indice	Argomento	Parametro formale
(1)	<code>ui</code>	<code>double</code>
(2)	<code>(ui - f)</code>	<code>float</code>
(3)	<code>0.2F</code>	<code>double</code>
(4)	<code>0</code>	<code>const float*</code>
(5)	<code>"stringa"</code>	<code>const char*</code>
(6)	<code>"stringa"</code>	<code>std::string</code>
(7)	<code>sc</code>	<code>int*</code>
(8)	<code>ui</code>	<code>const unsigned int&amp;</code>
(9)	<code>ai</code>	<code>int*</code>



4. (Domande a risposta aperta)

- (a) Fornire un esempio di violazione della ODR (One Definition Rule) che genera un errore *non* diagnosticabile in fase di compilazione in senso stretto (cioè prima della fase di collegamento).
- (b) I template di classe `std::vector` e `std::list` hanno interfacce simili, ma non identiche. Indicare almeno un metodo di `std::vector` che non è supportato da `std::list` e, viceversa, un metodo di `std::list` non supportato da `std::vector`.
- (c) Nella risoluzione dell'overloading, in base a quali criteri si stabilisce se una funzione candidata è o meno utilizzabile?
- (d) Spiegare brevemente l'utilità degli stream iterator, fornendo un semplice esempio di uso.
- (e) Enunciare il principio DIP (Dependency Inversion Principle).
- (f) Spiegare la differenza esistente tra un template di funzione ed una sua istanziazione.



## Esercizi per tutti

5. (Polimorfismo dinamico) Il codice seguente vorrebbe consentire la produzione di documentazione secondo diversi formati di stampa.

```
// File Generatore_Doc.hh
struct Generatore_Doc {
    virtual void stampa(const std::string& s) = 0;
    void link(const std::string& uri, const std::string& testo);
    virtual void grassetto(bool mode) = 0;
    virtual void corsivo(bool mode) = 0;
    ~Generatore_Doc() {}
};
```

```
// File Generatore_HTML.hh
#include "Generatore_Doc.hh"
struct Generatore_HTML : private Generatore_Doc {
    void stampa(const std::string& s);
    void grassetto(bool mode);
    void corsivo(bool mode);
};
```

```
// File Generatore_PDF.hh
#include "Generatore_Doc.hh"
```



```
// File Generatore_PDF.hh
#include "Generatore_Doc.hh"
struct Generatore_PDF : private Generatore_Doc {
    void stampa(const std::string& s);
    void grassetto(bool mode);
    void corsivo(bool mode);
    void salto_pagina();
};
```

```
// File Codice_Utente.cc
#include "Generatore_Doc.hh"
#include "Generatore_HTML.hh"
#include "Generatore_PDF.hh"
void codice_utente(Generatore_Doc* gdoc) {
    gdoc->hyperlink("http://www.unipr.it", "UNIPR");
    if (Generatore_PDF gpdf = dynamic_cast<Generatore_PDF>(gdoc))
        gpdf->salto_pagina();
    else
        gdoc->stampa("<HR>"); // Simulo il cambio pagina in HTML.
}
```

Individuare e correggere gli errori di sintassi, di semantica e di progettazione presenti nel codice mostrato. Nota Bene: va riscritto tutto il codice, usando i commenti per indicare la suddivisione del codice in file sorgente distinti.



6. Indicare l'output prodotto dal seguente programma.

```
#include <iostream>

struct A {
    A() { std::cout << "Ctor A::A()" << std::endl; }
    virtual void f(int) { std::cout << "A::f(int)" << std::endl; }
    virtual void f(double) { std::cout << "A::f(double)" << std::endl; }
    virtual void g(int) { std::cout << "A::g(int)" << std::endl; }
    virtual ~A() { std::cout << "Dtor A::~~A()" << std::endl; }
};

struct B : public A {
    B() { std::cout << "Ctor B::B()" << std::endl; }
    using A::f;
    virtual void f(char) { std::cout << "B::f(char)" << std::endl; }
    void g(int) { std::cout << "B::g(int)" << std::endl; }
    ~B() { std::cout << "Dtor B::~~B()" << std::endl; }
};

struct C : public B {
    virtual void f(char) { std::cout << "C::f(char)" << std::endl; }
    ~C() { std::cout << "Dtor C::~~C()" << std::endl; }
};
```



```
struct C : public B {  
    virtual void f(char) { std::cout << "C::f(char)" << std::endl; }  
    ~C() { std::cout << "Dtor C::~~C()" << std::endl; }  
};
```

```
int main() {  
    A a;  
    C c;  
    A& ra = a;  
    B* pb = &c;  
    std::cout << "=== 1 ===" << std::endl;  
    pb->f(1.0);  
    pb->f('a');  
    pb->g(3.1415);  
    std::cout << "=== 2 ===" << std::endl;  
    c.f(1);  
    c.f(1.0);  
    c.g(3.1415);  
    std::cout << "=== 3 ===" << std::endl;  
    a.f(1);  
    ra.f('a');  
    ra.g(1);  
    return 0;  
}
```