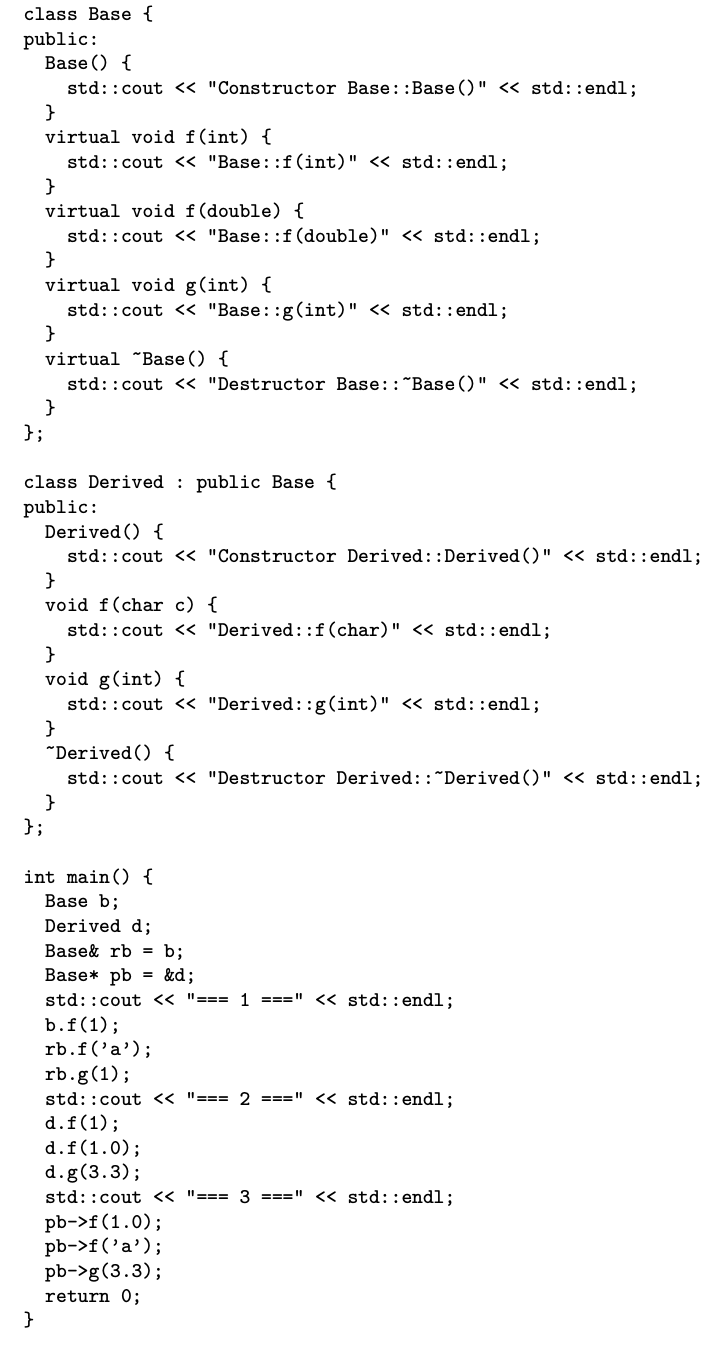
**METODOLOGIE**

**DI**  
**PROGRAMMAZIONE**

**(Output)**

01-02-2005



ConstructorBase::Base()

Constructor Base::Base()

Constructor Derived::Derived()

ConstructorBase::Base() *//I puntatori e le reference non attivano il costruttore, nel caso dei puntatori solo nel caso del comando new, avviene la chiamata al costruttore.*

=== 1 ===

Base::f(int)

Base::f(int)  *//conversione implicita, char int*

Base::g(int)

=== 2 ===

Derived::f(char)*//vado a vedere le funzioni di base solo se non è possibile utilizzare/convertire in derived* Derived::f(char)

Derived::g(int)

=== 3 ===

Base::f(double)

Derived::f(char)

Derived::g(int)

Destructor Derived::~Derived()

Destructor Base::~Base()

Destructor Base::~Base()

22-02-2005



**---Panda ying\_yang---**

Constructor ZooAnimal*//Prima le classi virtuali*

Constructor Edangered

Constructor Bear

Construction Raccon

Construction Panda

**---ying\_yang.print()---**

Panda::print

**---Bear b = ying\_yang---**

*//non succede nulla perché non creo nulla ma semplicemente copio*

---b.print()---

Bear::print

**--- ZooAnimal\* pz = &ying\_yang---**

*//il puntatore non attiva i costruttori, solo con il comando new*

**---pz->print()---**

Panda::print

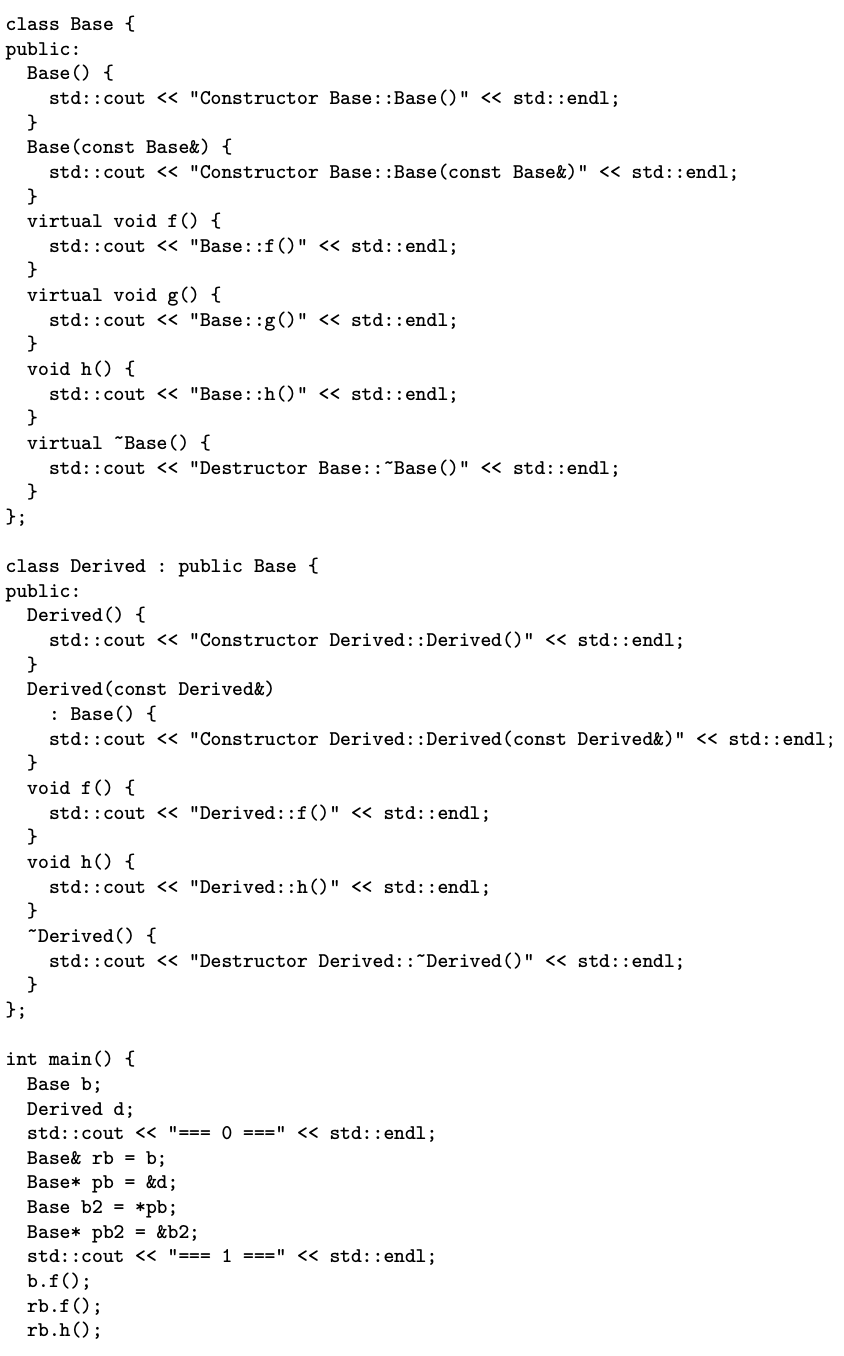
**---Endangered& re = ying\_yang---**

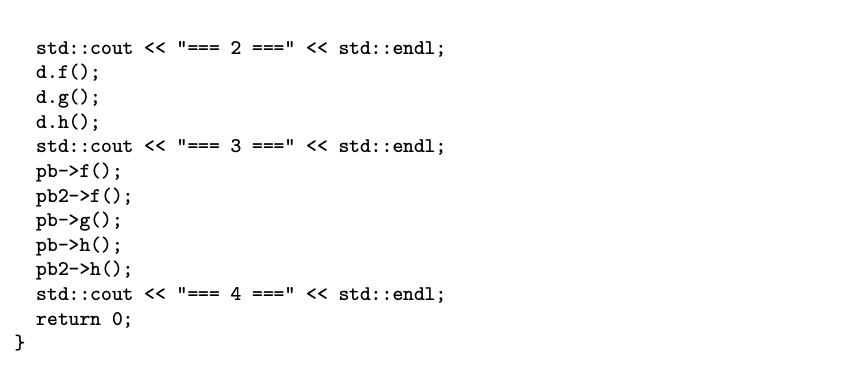
//la referenze non crea nulla

**---re.print()---**

Endangered::print*//referenziando un oggetto, cambio la classe mentre il puntatore non fa nulla*

20-09-2005





#Base b

Constructor Base::Base()

#Derived d

Constructor Base::Base()

Constructor Derived::Derived()

=== 0 ===

#Base b2 = \*pb //è come se stessi usando una reference

Constructor Base::Base(const Base&)

=== 1 ===

# b.f()

Base::f()

# rb.f()

Base::f()

#rb.h()

Base::h()

=== 2 ===

#d.f()

Derived::f()

#d.g()

Base::g()

#d.h()

Derived::h()

=== 3 ===

#pb->f()

Derived::f()

#pb2->f()

Base::f() //puntatore a puntatore

#pb->g()

Base::g()

#pb->h()

Base::h()

#pb2->h()

Base::h()

=== 4 ===

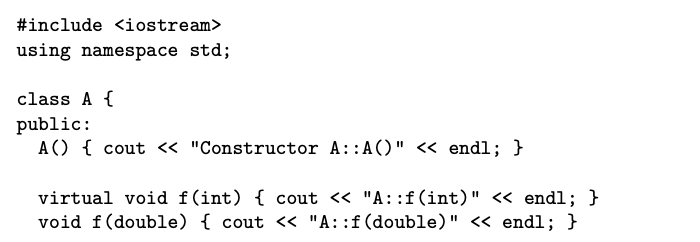
Destructor Base::~Base()

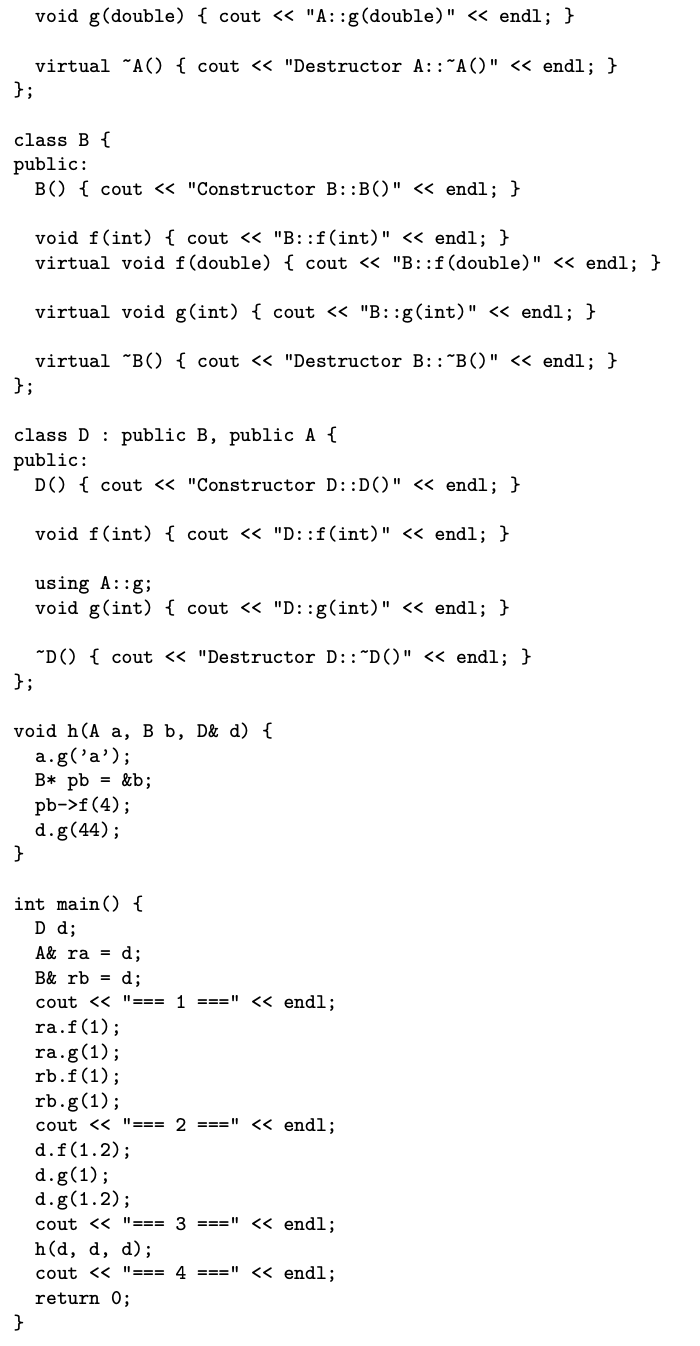
Destructor Derived::~Derived()

Destructor Base::~Base()

Destructor Base::~Base()

06-02-2006





Constructor B::B()

Constructor A::A()

Constructor D::D() //non viene creato più nulla perchè sono delle reference

=== 1 ===

D::f(int) //dovrei usare la classe A ma li c’è una virtual

A::g(double)

B::f(int)

D::g(double)

=== 2 ===

D::f(int)

D::g(int)

A::g(double)// con using vede entrambe le funzioni g e sceglie la migliore(conversione esatta)

=== 3 ===

// A e B sono creati ma come copia, manca il costruttore di copia nelle classi, per esempio *A(const A&){cout << "Constructor A::A(A)" << endl;}*

A::g(double)

B::f(int)

D::g(int)

Destructor A::~A()

Destructor B::~B() // A e B vengono eliminati perchè non sono stati passati per reference ma direttamente creati

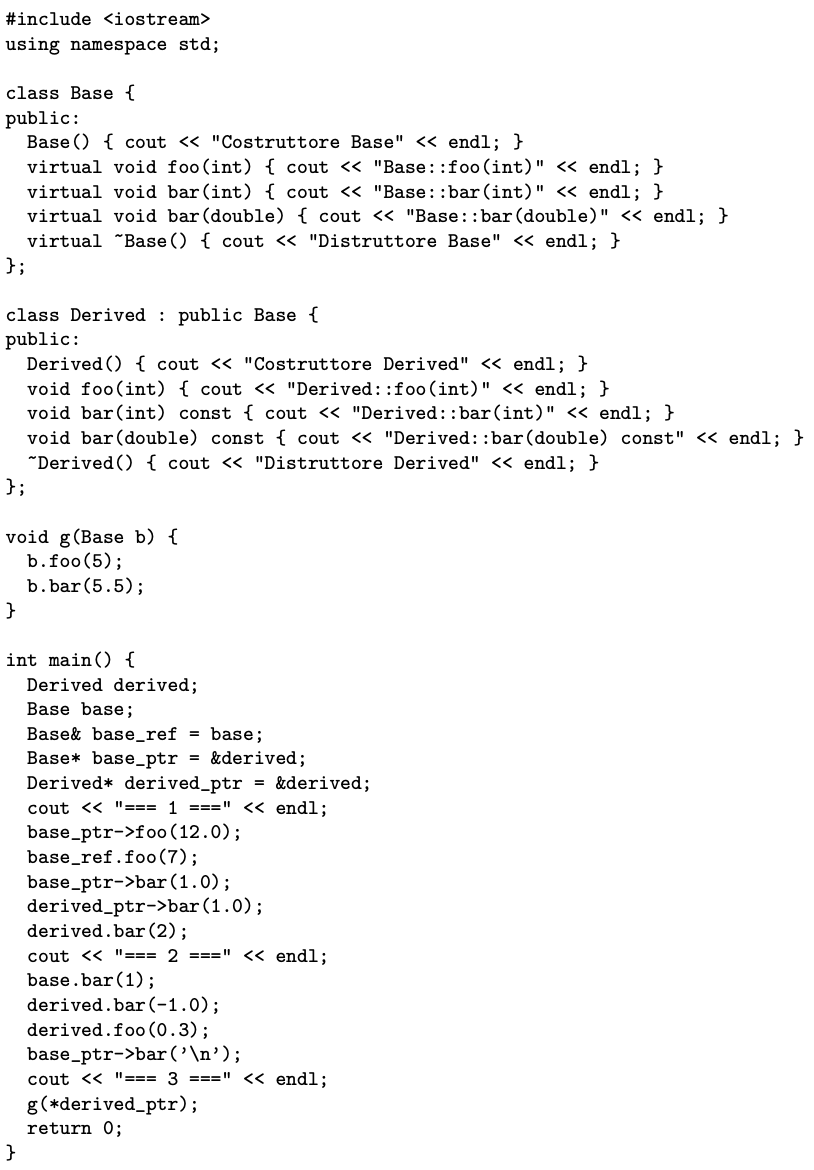
=== 4 ===

Destructor D::~D()

Destructor A::~A()

Destructor B::~B()

27-02-2006



Costruttore Base

Costruttore Derived

Costruttore Base

=== 1 ===

Derived :: foo(int)

Base :: foo(int)

Base :: bar(double)

Derived :: bar(double) const

Derived :: bar(int)

=== 2 ===

Base :: bar(int)

Derived :: bar(double) const

Derived :: foo(int)

Base :: bar(int) //Essendo Virtual, può richiamare anche la funzione in Derived, ma la funzione in Base è la migliore

=== 3 ===

Base :: foo(int)

Base :: bar(double)

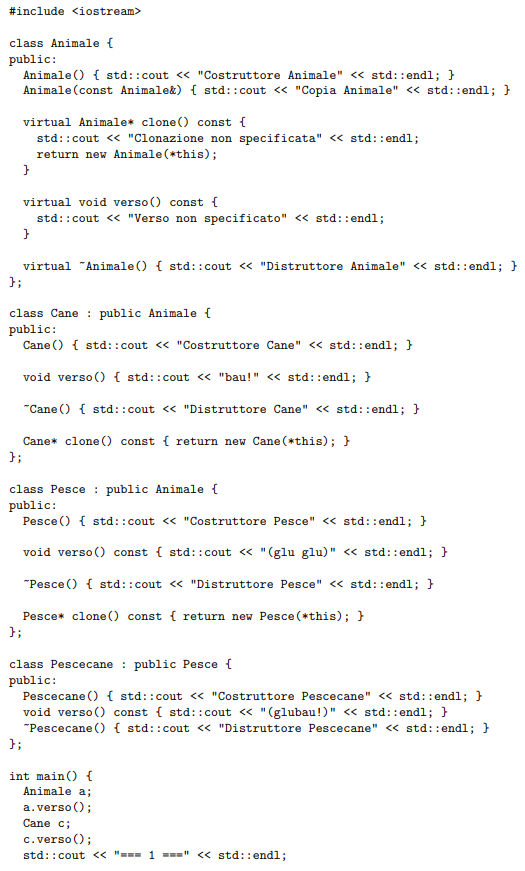
Distruttore Base

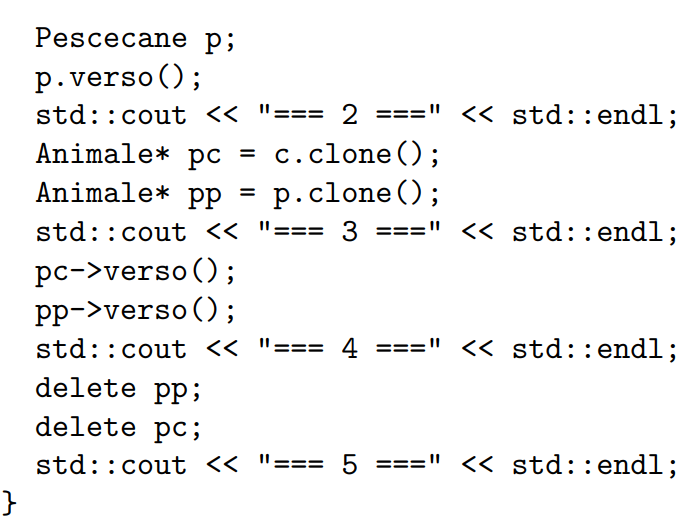
Distruttore Base

Distruttore Derived

Distruttore Base

16-06-2008





Costruttore Animale

Verso non specificato

Costruttore Animale

Costruttore Cane

Bau!

=== 1 ===

Costruttore Animale

Costruttore Pesce

Costruttore PesceCane

(glubau!)

=== 2 ===

Copia Animale

Copia Animale

=== 3 ===

Verso non specificato // stai attento al const(<3)

Glu glu

=== 4 ===

Distruttore Pesce

Distruttore Animale

Distruttore Cane

Distruttore Animale

=== 5 ===

Distruttore PesceCane

Distruttore Pesce

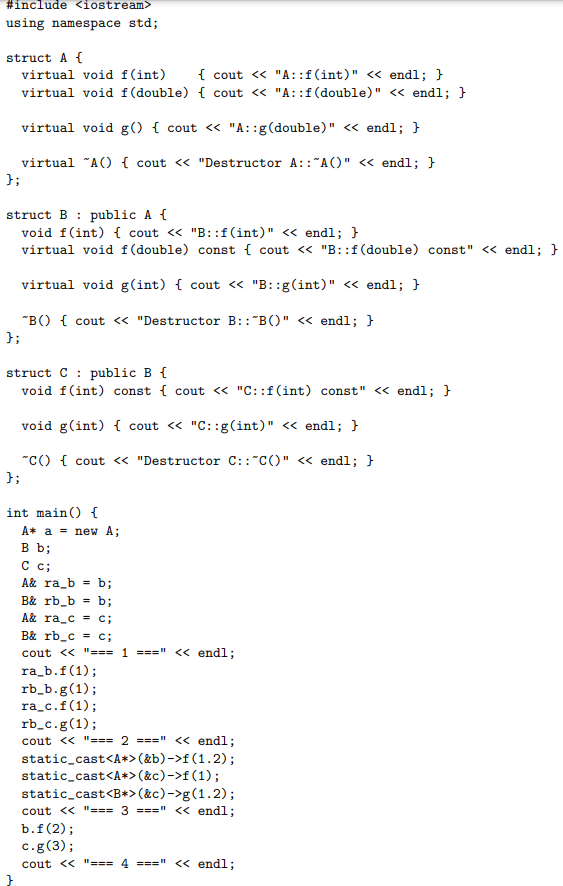
Distruttore Animale

Distruttore Cane

Distruttore Animale

Distruttore Animale

14-06-2010



=== 1 ===

B :: f(int)

B :: g(int)

B :: f(int)

C :: g(int)

=== 2 ===//controllo il const con il tipo migliore, partendo dal basso

A :: f(double)

B :: f(int)

C :: g(int)

=== 3 ===

B :: f(int)

C :: g(int)

=== 4 ===

Destructor C::-C()

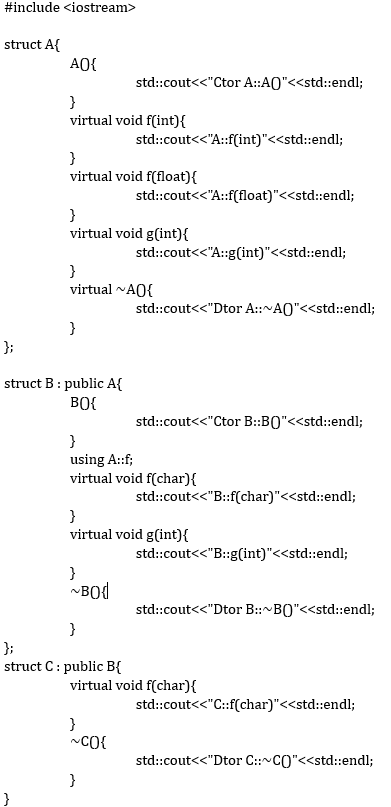
Destructor B::-B()

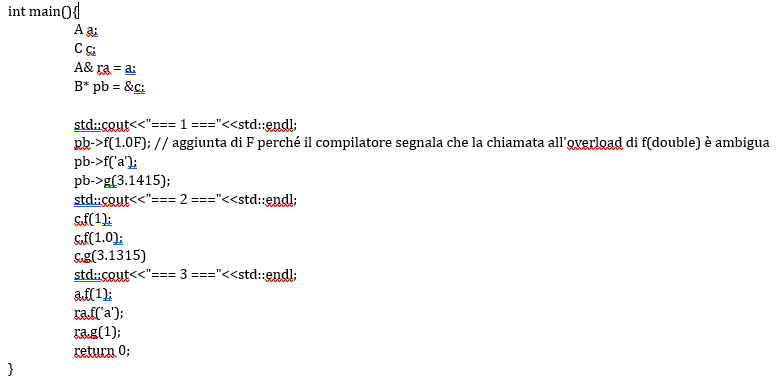
Destructor A::-A()

Destructor B::-B()

Destructor A::-A()

05-07-2022





Ctor A::A()

Ctor A::A()

Ctor B::B()

=== 1 ===

A::f(float)

C::f(char)

B::g(int)

=== 2 ===

C::f(char)

C::f(char)

B::g(int)

=== 3 ===

A::f(int)

A::f(int)

A::g(int)

=== 4 ===

Dtor C::-C()

Dtor B::-B()

Dtor A::-A()

Dtor A::-A()