

Nome..... Cognome..... Matricola.....

Esercizio Funzione

Si richiamano i seguenti fatti concernenti la libreria di I/O standard.

- `ios` è la classe base astratta e virtuale della gerarchia di tipi della libreria di I/O; la classe `istream` è derivata direttamente e virtualmente da `ios`; la classe `ifstream` è derivata direttamente da `istream`.
- `ios` rende disponibile un metodo costante e non virtuale `bool fail()` con il seguente comportamento: una invocazione `s.fail()` ritorna `true` se e solamente se lo stream `s` è in uno stato di fallimento (cioè, il failbit di `s` vale 1).
- `istream` rende disponibile un metodo non costante e non virtuale `long tellg()` con il seguente comportamento: una invocazione `s.tellg()`:
 1. se `s` è in uno stato di fallimento allora ritorna -1;
 2. altrimenti, cioè se `s` non è in uno stato di fallimento, ritorna la posizione della testina di input di `s`.
- `ifstream` rende disponibile un metodo costante e non virtuale `bool is_open()` con il seguente comportamento: una invocazione `s.is_open()` ritorna `true` se e solo se il file associato allo stream `s` è aperto.

Definire una funzione `long Fun(const ios&)` con il seguente comportamento: una invocazione `Fun(s)`:

- (1) se `s` è in uno stato di fallimento lancia una eccezione di tipo `Fallimento`, dove la classe `Fallimento` va esplicitamente definita;
- (2) se `s` non è in uno stato di fallimento allora:
 - (a) se `s` non è un `ifstream` ritorna -2;
 - (b) se `s` è un `ifstream` ed il file associato non è aperto ritorna -1;
 - (c) se `s` è un `ifstream` ed il file associato è aperto ritorna la posizione della cella corrente di input di `s`.

SOLUZIONE

Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout<<"B::f "; g(); j();}
    virtual void g() const {cout<<" B::g ";}
    virtual const B* j() {cout<<" B::j "; return this;}
    virtual void k() {cout<<" B::k "; j(); m();}
    void m() {cout<<" B::m "; g(); j();}
    virtual B& n() {cout<<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout<<" C::g ";}
    void k() override {cout<<" C::k "; B::n();}
    virtual void m() {cout<<" C::m "; g(); j();}
    B& n() override {cout<<" C::n "; return *this;}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout<<" E::g ";}
    const E* j() {cout<<" E::j "; return this;}
    void m() {cout<<" E::m "; g(); j();}
    D& n() final {cout<<" E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

$p4 \rightarrow C/G$
 $p4 \rightarrow F()$



```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout<<" D::g ";}
    const B* j() {cout<<" D::j "; return this;}
    void k() const {cout<<" D::k "; k();}
    void m() {cout<<" D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout<<" F::k "; g();}
    void m() {cout<<" F::m "; j();}
};
```

Queste definizioni compilano correttamente (con opportuni `#include` e `using`). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D&>(p3->n()))<g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())<g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())<k(); .....
(dynamic_cast<const F*>(p1->j()))<g(); .....
(dynamic_cast<E*>(p6))<j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))<k(); .....
delete p7; .....
```

Override = Annotazione a te stesso per dire "sono un sottometodo"

Final = Non va più giù di lui l'overriding

Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout<< " B::f "; g(); j();}
    virtual void g() const {cout<< " B::g ";}
    virtual const B* j() {cout<< " B::j "; return this;}
    virtual void k() {cout<< " B::k "; j(); m();}
    void m() {cout<< " B::m "; g(); j();}
    virtual B& n() {cout<< " B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout<< " C::g ";}
    void k() override {cout<< " C::k "; B::n();}
    virtual void m() {cout<< " C::m "; g(); j();}
    B& n() override {cout<< " C::n "; return *this;}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout<< " E::g ";}
    const E* j() {cout<< " E::j "; return this;}
    void m() {cout<< " E::m "; g(); j();}
    D& n() final {cout<< " E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

(p4->n()).m();

$p4 = C/E$



```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout<< " D::g ";}
    const B* j() {cout<< " D::j "; return this;}
    void k() const {cout<< " D::k "; k();}
    void m() {cout<< " D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout<< " F::k "; g();}
    void m() {cout<< " F::m "; j();}
};
```

Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D*>(p3->n()))>g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())>g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())>k(); .....
(dynamic_cast<const F*>(p1->j()))>g(); .....
(dynamic_cast<E*>(p6))>j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```

Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout <<" B::f "; g(); j();}
    virtual void g() const {cout <<" B::g ";}
    virtual const B* j() {cout<<" B::j "; return this;}
    virtual void k() {cout <<" B::k "; j(); m(); }
    void m() {cout <<" B::m "; g(); j();}
    virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout <<" C::g ";}
    void k() override {cout <<" C::k "; B::n();}
    virtual void m() {cout <<" C::m "; g(); j();}
    B& n() override {cout <<" C::n "; return *this;}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout <<" E::g ";}
    const E* j() {cout <<" E::j "; return this;}
    void m() {cout <<" E::m "; g(); j();}
    D& n() final {cout <<" E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

(p5->n()).g();

CONST B / D

THIS NON CONST / B LO È

NC = NON COMPILA

```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout <<" D::g ";}
    const B* j() {cout <<" D::j "; return this;}
    void k() const {cout <<" D::k "; k();}
    void m() {cout <<" D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout <<" F::k "; g();}
    void m() {cout <<" F::m "; j();}
};
```

Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D*>(p3->n())) .g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())->g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```

Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout <<" B::f "; g(); j();}
    virtual void g() const {cout <<" B::g ";}
    virtual const B* j() {cout<<" B::j "; return this;}
    virtual void k() {cout <<" B::k "; j(); m(); }
    void m() {cout <<" B::m "; g(); j();}
    virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout <<" C::g ";}
    void k() override {cout <<" C::k "; B::n();}
    virtual void m() {cout <<" C::m "; g(); j();}
    B& n() override {cout <<" C::n "; return *this;}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout <<" E::g ";}
    const E* j() {cout <<" E::j "; return this;}
    void m() {cout <<" E::m "; g(); j();}
    D& n() final {cout <<" E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout <<" D::g ";}
    const B* j() {cout <<" D::j "; return this;}
    void k() const {cout <<" D::k "; k();}
    void m() {cout <<" D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout <<" F::k "; g();}
    void m() {cout <<" F::m "; j();}
};
```

Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D*>(p3->n()))>.g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())>->g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())>->k(); .....
(dynamic_cast<const F*>(p1->j()))>->g(); .....
(dynamic_cast<E*>(p6))>->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>->k(); .....
delete p7; .....
```

STACK OVERFLOW
= CHARACTERS IN PRINTS

BRT

PH → k()

k() k()
k() - - -

VIRTUAL k() { ... k(); }

Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout <<" B::f "; g(); j();}
    virtual void g() const {cout <<" B::g ";}
    virtual const B* j() {cout<<" B::j "; return this;}
    virtual void k() {cout <<" B::k "; j(); m();}
    void m() {cout <<" B::m "; g(); j();}
    virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout <<" C::g ";}
    void k() override {cout <<" C::k "; B::n();}
    virtual void m() {cout <<" C::m "; g(); j();}
    B& n() override {cout <<" C::n "; return *this;}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout <<" E::g ";}
    const E* j() {cout <<" E::j "; return this;}
    void m() {cout <<" E::m "; g(); j();}
    D& n() final {cout <<" E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

(dynamic_cast<const F*>(p1->j()))->g();

p1 -> B/E

DOPO THIS => CONST F

p1 =

NON FOI

```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout <<" D::g ";}
    const B* j() {cout <<" D::j "; return this;}
    void k() const {cout <<" D::k "; k();}
    void m() {cout <<" D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout <<" F::k "; g();}
    void m() {cout <<" F::m "; j();}
};
```

Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D*>(p3->n())) .g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())->g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```

Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout <<" B::f "; g(); j();}
    virtual void g() const {cout <<" B::g ";}
    virtual const B* j() {cout<<" B::j "; return this;}
    virtual void k() {cout <<" B::k "; j(); m(); }
    void m() {cout <<" B::m "; g(); j();}
    virtual B& n() {cout <<" B::n "; return *this;}
};
```

$(\text{dynamic_cast}<\text{C}*>(\text{const_cast}<\text{B}*>(\text{p7}))) \rightarrow \text{k}();$

CONST B p7 = FC;
C p7 = FC;

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout <<" C::g ";}
    void k() override {cout <<" C::k "; B::n();}
    virtual void m() {cout <<" C::m "; g(); j();}
    B& n() override {cout <<" C::n "; return *this;}
};
```

```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout <<" D::g ";}
    const B* j() {cout <<" D::j "; return this;}
    void k() const {cout <<" D::k "; k();}
    void m() {cout <<" D::m "; g(); j();}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout <<" E::g ";}
    const E* j() {cout <<" E::j "; return this;}
    void m() {cout <<" E::m "; g(); j();}
    D& n() final {cout <<" E::n "; return *this;}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout <<" F::k "; g();}
    void m() {cout <<" F::m "; j();}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

Queste definizioni compilano correttamente (con opportuni `#include` e `using`). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D&>(p3->n())) .g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())->g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```

Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout <<" B::f "; g(); j();}
    virtual void g() const {cout <<" B::g ";}
    virtual const B* j() {cout<<" B::j "; return this;}
    virtual void k() {cout <<" B::k "; j(); m(); }
    void m() {cout <<" B::m "; g(); j();}
    virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout <<" C::g ";}
    void k() override {cout <<" C::k "; B::n();}
    virtual void m() {cout <<" C::m "; g(); j();}
    B& n() override {cout <<" C::n "; return *this;}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout <<" E::g ";}
    const E* j() {cout <<" E::j "; return this;}
    void m() {cout <<" E::m "; g(); j();}
    D& n() final {cout <<" E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

NON CONST
(p1->j())->k(); → NON
COMPILA

PER COMPILARE → CONST_CAST
→ TOLGO CONST
DA TUTTO
A UNO
PIÙ

```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout <<" D::g ";}
    const B* j() {cout <<" D::j "; return this;}
    void k() const {cout <<" D::k "; k();}
    void m() {cout <<" D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout <<" F::k "; g();}
    void m() {cout <<" F::m "; j();}
};
```

THIS NON
CONST
↓
RITORNA
A
CONST

→ NON COMPILA

Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D&>(p3->n())) .g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())->g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```


Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout <<" B::f "; g(); j();}
    virtual void g() const {cout <<" B::g ";}
    virtual const B* j() {cout<<" B::j "; return this;}
    virtual void k() {cout <<" B::k "; j(); m(); }
    void m() {cout <<" B::m "; g(); j();}
    virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout <<" C::g ";}
    void k() override {cout <<" C::k "; B::n();}
    virtual void m() {cout <<" C::m "; g(); j();}
    B& n() override {cout <<" C::n "; return *this;}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout <<" E::g ";}
    const E* j() {cout <<" E::j "; return this;}
    void m() {cout <<" E::m "; g(); j();}
    D& n() final {cout <<" E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

$[C^* p = \text{new } F(f);]$
Copia

C() D() E() Fc

```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout <<" D::g ";}
    const B* j() {cout <<" D::j "; return this;}
    void k() const {cout <<" D::k "; k();}
    void m() {cout <<" D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout <<" F::k "; g();}
    void m() {cout <<" F::m "; j();}
};
```



Copia

Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D&>(p3->n()))<g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())<g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())<k(); .....
(dynamic_cast<const F*>(p1->j()))<g(); .....
(dynamic_cast<E*>(p6))<j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))<k(); .....
delete p7; .....
```

Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout<< " B::f "; g(); j();}
    virtual void g() const {cout<< " B::g ";}
    virtual const B* j() {cout<< " B::j "; return this;}
    virtual void k() {cout<< " B::k "; j(); m();}
    void m() {cout<< " B::m "; g(); j();}
    virtual B& n() {cout<< " B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout<< " C::g ";}
    void k() override {cout<< " C::k "; B::n();}
    virtual void m() {cout<< " C::m "; g(); j();}
    B& n() override {cout<< " C::n "; return *this;}
};
```

SE NON PRACATO = NO STAMPA

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout<< " E::g ";}
    const E* j() {cout<< " E::j "; return this;}
    void m() {cout<< " E::m "; g(); j();}
    D& n() final {cout<< " E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

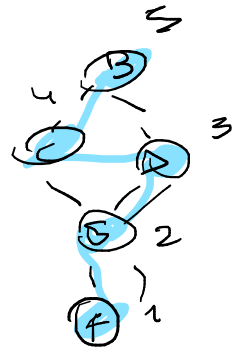
delete p7;

CONST B / F
(B)

~F ~D ~E ~B

```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout<< " D::g ";}
    const B* j() {cout<< " D::j "; return this;}
    void k() const {cout<< " D::k "; k();}
    void m() {cout<< " D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout<< " F::k "; g();}
    void m() {cout<< " F::m "; j();}
};
```



Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D*>(p3->n()))>g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())>g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())>k(); .....
(dynamic_cast<const F*>(p1->j()))>g(); .....
(dynamic_cast<E*>(p6))>j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```

Esercizio Cosa Stampa

```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout <<" B::f "; g(); j();}
    virtual void g() const {cout <<" B::g ";}
    virtual const B* j() {cout<<" B::j "; return this;}
    virtual void k() {cout <<" B::k "; j(); m(); }
    void m() {cout <<" B::m "; g(); j();}
    virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout <<" C::g ";}
    void k() override {cout <<" C::k "; B::n();}
    virtual void m() {cout <<" C::m "; g(); j();}
    B& n() override {cout <<" C::n "; return *this;}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout <<" E::g ";}
    const E* j() {cout <<" E::j "; return this;}
    void m() {cout <<" E::m "; g(); j();}
    D& n() final {cout <<" E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

Undefined behavior = Non definita dallo standard
= Vai a fare dei cast tra sottotipi che sono nulli

Errore runtime = La stampa va male durante l'esecuzione
0 come prima (stampa E::j, "ci pensa" poi dà errore)
OPPURE

La chiamata a k() ricorsivo

Non compila = Const correctness

- 1) Il puntatore const e ritorna una roba non cont
- 2) Viceversa (Puntatore non const e metodo const)

```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout <<" D::g ";}
    const B* j() {cout <<" D::j "; return this;}
    void k() const {cout <<" D::k "; k();}
    void m() {cout <<" D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout <<" F::k "; g();}
    void m() {cout <<" F::m "; j();}
};
```

Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....

(p4->n()).m(); .....

p3->k(); .....

(p3->n()).m(); .....

(dynamic_cast<D*>(p3->n()))>.g(); .....

p2->f(); .....

p2->m(); .....

(p2->j())->g(); .....

(p5->n()).g(); .....

F x; .....

C* p = new F(f); .....

p1->m(); .....

(p1->j())->k(); .....

(dynamic_cast<const F*>(p1->j()))->g(); .....

(dynamic_cast<E*>(p6))->j(); .....

(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....

delete p7; .....
```

Esercizio Cosa Stampa

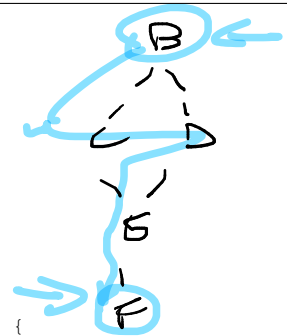
```
class B {
public:
    B() {cout<< " B() ";}
    virtual ~B() {cout<< " ~B() ";}
    virtual void f() {cout <<" B::f "; g(); j();}
    virtual void g() const {cout <<" B::g ";}
    virtual const B* j() {cout<<" B::j "; return this;}
    virtual void k() {cout <<" B::k "; j(); m(); }
    void m() {cout <<" B::m "; g(); j();}
    virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout <<" C::g ";}
    void k() override {cout <<" C::k "; B::n();}
    virtual void m() {cout <<" C::m "; g(); j();}
    B& n() override {cout <<" C::n "; return *this;}
};
```

```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout <<" E::g ";}
    const E* j() {cout <<" E::j "; return this;}
    void m() {cout <<" E::m "; g(); j();}
    D& n() final {cout <<" E::n "; return *this;}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

F x;



```
class D: virtual public B {
public:
    D() {cout<< " D() ";}
    ~D() {cout<< " ~D() ";}
    virtual void g() {cout <<" D::g ";}
    const B* j() {cout <<" D::j "; return this;}
    void k() const {cout <<" D::k "; k();}
    void m() {cout <<" D::m "; g(); j();}
};
```

```
class F: public E {
public:
    F() {cout<< " F() ";}
    ~F() {cout<< " ~F() ";}
    F(const F& x): B(x) {cout<< " Fc ";}
    void k() {cout <<" F::k "; g();}
    void m() {cout <<" F::m "; j();}
};
```

Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f(); .....
(p4->n()).m(); .....
p3->k(); .....
(p3->n()).m(); .....
(dynamic_cast<D&>(p3->n())) .g(); .....
p2->f(); .....
p2->m(); .....
(p2->j())->g(); .....
(p5->n()).g(); .....
F x; .....
C* p = new F(f); .....
p1->m(); .....
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```